▼ AMSTRAD ▼ SPECTRUM ▼ COMMODORE ▼ IBM ▼ MSX ▼

Informática y programación 37

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Informática y programación

37

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Una publicación de

EDICIONES SIGLO CULTURAL, S.A.

Director-editor:
RICARDO ESPAÑOL CRESPO.

Gerente:
ANTONIO G. CUERPO.

Directora de producción:
MARIA LUISA SUAREZ PEREZ.

Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación y Licenciado en Informática.
JOSE ARTECHE, Ingeniero de Telecomunicación.

Diseño y maquetación:
BRAVO-LOFISH.

Fotografía:
EQUIPO GALATA.

Dibujos:
JOSE OCHOA

TECNICAS DE PROGRAMACION: Manuel Alfonseca, Doctor Ingeniero de Telecomunicación y Licenciado en Informática. TECNICAS DE ANALISIS: José Arteche, Ingeniero en Telecomunicación. LENGUAJE MAQUINA 8086: Juan Rojas, Licenciado en Ciencias Físicas e Ingeniero Industrial. PASCAL: Juan Ignacio Puyol, Ingeniero Industrial. PROGRAMAS (educativos, de utilidad, de gestión y de juegos): Francisco Morales, Técnico en Informática y colaboradores. Coordinador de Aula de Informática Aplicada (AIA): Alejandro Marcos, Licenciado en Ciencias Químicas. BASIC: Esther Maldonado, Diplomada en Arquitectura. INFORMATICA BASICA: Virginia Muñoz, Diplomada en Informática. LENGUAJE MAQUINA Z-80: Joaquín Salvachúa, Diplomado en Telecomunicación y José Luis Tojo, Diplomado en Telecomunicación. LENGUAJE MAQUINA 6502: (desde el tomo 5): Juan José Gómez, Licenciado en Química. LOGO: Cristina Manzanera, Licenciada en Informática. APLICACIONES: Sociedad Tamariz, Diplomada en Telecomunicación. OTROS LENGUAJES (COBOL): Eloy Pérez, Licenciado en Informática. Ana Pastor, Licenciada en Informática.

Ediciones Siglo Cultural, S.A.

Dirección, redacción y administración:
Pedro Teixeira, 8, 2.ª planta. Teléf. 455 09 99. 28020 Madrid.

Publicidad:
Gofar Publicidad, S.A. Benito de Castro, 12 bis. 28028 Madrid.

Distribución en España:
COEDIS, S.A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.

Distribución en Ecuador: Muñoz Hnos.

Distribución en Perú: DISELPESA.

Distribución en Chile: Alfa Ltda.

Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América, 1532. Teléf.: 21 24 64.
Buenos Aires - 1.290. Argentina.

ISBN del tomo: 84-7688-189-4
ISBN de la obra: 84-7688-068-7

Fotocomposición:
ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.

Imprime:
MATEU CROMO. Pinto (Madrid).

Depósito legal: M. 5.677-1987
Printed in Spain - Impreso en España.

Suscripciones y números atrasados:
Ediciones Siglo Cultural, S.A.
Pedro Teixeira, 8, 2.ª planta. Teléf. 259 73 31. 28020 Madrid.

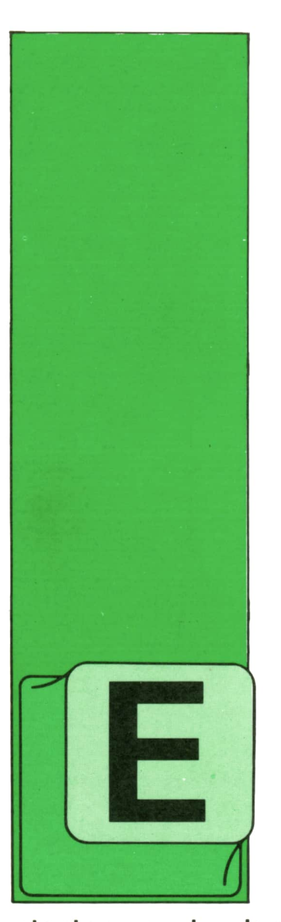
Abril, 1988.
P.V.P. Canarias: 335,–.

INDICE

4 BASIC
9 MAQUINA 8088
12 PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS
24 TECNICAS DE ANALISIS
27 TECNICAS DE PROGRAMACION
31 LOGO
34 PASCAL
38 OTROS LENGUAJES

BASIC

SONIDO

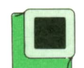




Sonido

EL lenguaje BASIC pone a nuestra disposición la posibilidad de crear infinidad de programas sonoros, desde sencillos efectos especiales como trinos, campanas, explosiones, etc., hasta las más sofisticadas composiciones musicales.

Sin embargo, no existe una instrucción BASIC única para la creación de sonidos en los distintos ordenadores. Las instrucciones más comunes son BEEP y SOUND, cada una de ellas con su formato particular; por tanto, vamos a estudiarlas una a una.



Beep

El SPECTRUM utiliza la instrucción BEEP para la generación de sonidos. Su formato es el siguiente:

BEEP D, T

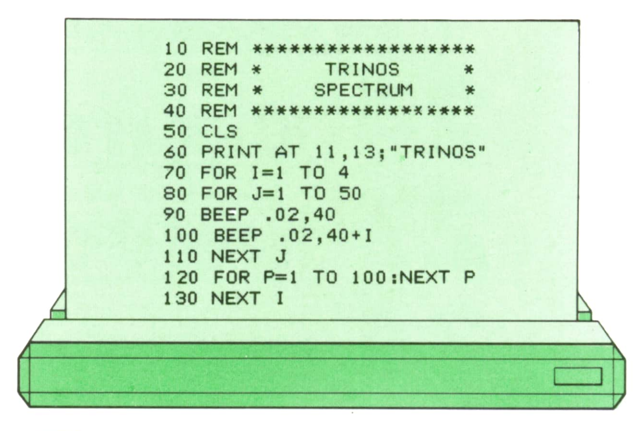
siendo D la duración del sonido en segundos y T el tono, expresado por un número entero comprendido entre −59 y +69. Cuanto menor es el número del tono, más grave es el sonido emitido y, a medida que aumentemos este parámetro, el tono irá siendo más agudo.

Para los iniciados en música diremos que el Do medio del piano corresponde al tono cero y la diferencia entre dos números consecutivos es de un semitono.

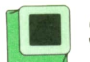
El programa 1 interpreta una escala en Do mayor en el SPECTRUM.

```
10 REM ************************
20 REM *  ESCALA EN DO MAYOR  *
30 REM *       SPECTRUM       *
40 REM ************************
50 CLS
60 FOR I=1 TO 8
70 READ T,N$
80 PRINT AT 11,15;N$;" "
90 BEEP .5,T
100 FOR P=1 TO 100:NEXT P
110 NEXT I
120 DATA 0,"DO",2,"RE",4,"MI",5,"FA"
130 DATA 7,"SOL",9,"LA",11,"SI",12,"DO"
```

Los no iniciados en el lenguaje musical pueden probar algún efecto especial, como los trinos que simula el programa 2.



```
10 REM *****************
20 REM *    TRINOS     *
30 REM *   SPECTRUM    *
40 REM *****************
50 CLS
60 PRINT AT 11,13;"TRINOS"
70 FOR I=1 TO 4
80 FOR J=1 TO 50
90 BEEP .02,40
100 BEEP .02,40+I
110 NEXT J
120 FOR P=1 TO 100:NEXT P
130 NEXT I
```



Sound

La instrucción SOUND se utiliza en varios ordenadores, aunque con formatos muy diferentes; por tanto, vamos a ir por partes.

En el AMSTRAD la construcción SOUND tiene el siguiente formato:

SOUND C,T,D,V,EV,ET,R

De estos siete parámetros sólo son imprescindibles los dos primeros, mientras que los otros cinco son opcionales.

El parámetro C permite seleccionar los canales de sonido, ya que el AMSTRAD dispone de tres canales distintos. C puede ser cualquier número entero entre 1 y 255, ya que es posible hacer 255 combinaciones distintas de los canales.

El parámetro T define el tono del sonido. Debe ser un número entero comprendido entre 0 y 4095. Cuanto menor sea el número, más agudo será el sonido. El Do medio del piano se corresponde con el tono 478.

El parámetro D nos permite especificar la duración del sonido en centésimas de segundo. Podemos darle cualquier valor comprendido entre 1 y 32767.

El parámetro V permite definir el volumen del sonido con un número entero comprendido entre 0 (mínimo) y 15 (máximo).

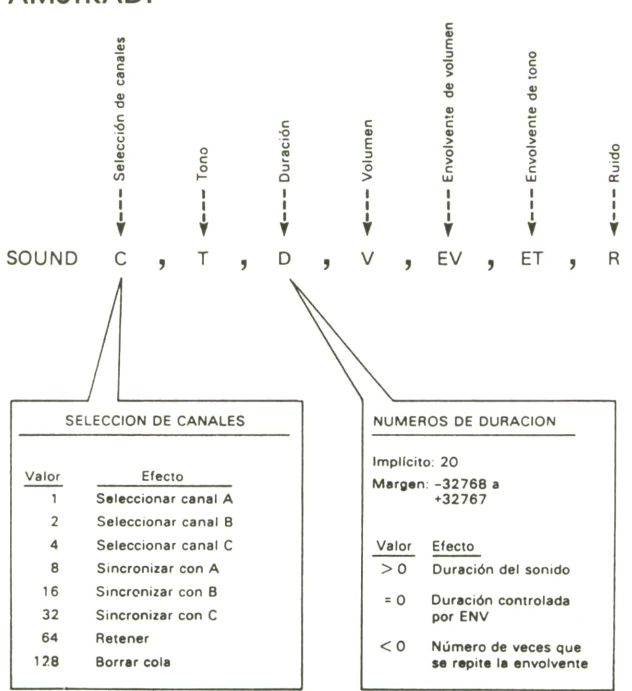
El programa 3 es un ejemplo de funcionamiento de estos cuatro parámetros. El ordenador interpreta una escala en Do mayor ascendente en la que cada nota dura el doble que la anterior, además de ir aumentando el volumen. A continuación suena la misma escala en sentido descendente, disminuyendo progresivamente la duración y el volumen.

```
10 REM *******************
20 REM *      ESCALAS     *
30 REM *      AMSTRAD     *
40 REM *******************
50 CLS
60 LET D=2:LET A=2
70 FOR V=1 TO 15 STEP 2
80 GOSUB 150
90 NEXT V
100 LET A=1/2
110 FOR V=15 TO 1 STEP -2
120 GOSUB 150
130 NEXT V
140 END
150 READ T,N$
160 LOCATE 16,8:PRINT "NOTA: ";N$;" "
170 LOCATE 16,12:PRINT "DURACION: ";D;"
180 LOCATE 16,16:PRINT "VOLUMEN: ";V;" "
190 SOUND 1,T,D,V
200 LET D=D*A
210 FOR P=1 TO 500:NEXT P
220 RETURN
230 DATA 239,DO,213,RE,190,MI,179,FA
240 DATA 159,SOL,142,LA,127,SI,119,DO
250 DATA 119,DO,127,SI,142,LA,159,SOL
260 DATA 179,FA,190,MI,213,RE,239,DO
```

EV es la envolvente de volumen que permite variar el volumen de un sonido mientras está sonando. Análogamente, ET es la envolvente de tono y permite variar el tono de un sonido mientras esté sonando.

Finalmente, el parámetro R permite la generación de ruidos.

En la figura 1 podemos ver resumidas todas las características de SOUND en el AMSTRAD.





Los parámetros de SOUND en el AMSTRAD.

El parámetro de generación de ruido que admite la instrucción SOUND en el AMSTRAD permite la creación de efectos especiales tan interesantes como el simulado en el programa 4, que produce el efecto de caída de una bomba y su explosión posterior.

```
10 REM **********************
20 REM *   CAIDA DE BOMBA    *
30 REM *       AMSTRAD       *
40 REM **********************
50 CLS
60 LOCATE 19,13:PRINT "CAIDA"
70 FOR I=50 TO 150
80 SOUND 1,I,3,15,0,0,0
90 NEXT I
100 LOCATE 16,13:PRINT "EXPLOSION"
110 FOR I=1 TO 100
120 SOUND 1,1,3,15,0,0,31
130 NEXT I
```

En cuanto al IBM, también dispone de la instrucción SOUND, aunque con un formato mucho más sencillo:

SOUND F,D

siendo el parámetro F la frecuencia del sonido expresada en hertzios. Puede tomar cualquier valor numérico comprendido entre 37 y 32767.

El parámetro D indica la duración del sonido en tic-tacs de reloj, sabiendo que un segundo son 18,2 tic-tacs. La duración puede ser cualquier expresión numérica comprendida entre 0 y 65535.

En la figura 2 podemos ver las frecuencias (para IBM) y los períodos de tono (para AMSTRAD) correspondientes a las ocho octavas utilizadas normalmente en música.

NOTA	FRECUENCIA	PERIODO	ERROR RELATIVO	
C DO	32.703	3822	-Ø.ØØ7%	
C# DO#	34.648	3608	+Ø.ØØ7%	
D RE	36.708	3405	-Ø.ØØ7%	
D# RE#	38.891	3214	-Ø.ØØ4%	
E MI	41.203	3034	+Ø.ØØ9%	
F FA	43.654	2863	-Ø.Ø16%	
F# FA#	46.249	2703	+Ø.ØØ9%	Octava -3
G SOL	48.999	2551	-Ø.ØØ2%	
G# SOL#	51.913	2408	+Ø.ØØ5%	
A LA	55.ØØØ	2273	+Ø.Ø12%	
A# LA#	58.27Ø	2145	-Ø.ØØ8%	
B SI	61.735	2Ø25	+Ø.Ø11%	

NOTA	FRECUENCIA	PERIODO	ERROR RELATIVO	
C DO	65.4Ø6	1911	-Ø.ØØ7%	
C# DO#	69.296	18Ø4	+Ø.ØØ7%	
D RE	73.416	17Ø3	+Ø.Ø22%	
D# RE#	77.782	16Ø7	-Ø.ØØ4%	
E MI	82.4Ø7	1517	+Ø.ØØ9%	
F FA	87.3Ø7	1432	+Ø.Ø19%	
F# FA#	92.499	1351	-Ø.Ø28%	Octava -2
G SOL	97.999	1276	+Ø.Ø37%	
G# SOL#	1Ø3.826	12Ø4	+Ø.ØØ5%	
A LA	11Ø.ØØØ	1136	-Ø.Ø32%	
A# LA#	116.541	1Ø73	+Ø.Ø39%	
B SI	123.471	1Ø12	-Ø.Ø38%	

NOTA	FRECUENCIA	PERIODO	ERROR RELATIVO	
C DO	13Ø.813	956	+Ø.Ø46%	
C# DO#	138.591	9Ø2	+Ø.ØØ7%	
D RE	146.832	851	-Ø.Ø37%	
D# RE#	155.564	8Ø4	+Ø.Ø58%	
E MI	164.814	758	-Ø.Ø57%	
F FA	174.614	716	+Ø.Ø19%	
F# FA#	184.997	676	+Ø.Ø46%	Octava -1
G SOL	195.998	638	+Ø.Ø37%	
G# SOL#	2Ø7.652	6Ø2	+Ø.ØØ5%	
A LA	22Ø.ØØØ	568	-Ø.Ø32%	
A# LA#	233.Ø82	536	-Ø.Ø55%	
B SI	246.942	5Ø6	-Ø.Ø38%	

NOTA	FRECUENCIA	PERIODO	ERROR RELATIVO	
C DO	261.626	478	+Ø.Ø46%	DO media
C# DO#	277.183	451	+Ø.ØØ7%	
D RE	293.665	426	+Ø.Ø81%	
D# RE#	311.127	4Ø2	+Ø.Ø58%	
E MI	329.628	379	-Ø.Ø57%	
F FA	349.228	358	+Ø.Ø19%	
F# FA#	369.994	338	+Ø.Ø46%	Octava Ø
G SOL	391.995	319	+Ø.Ø37%	
G# SOL#	415.3Ø5	3Ø1	+Ø.ØØ5%	
A LA	44Ø.ØØØ	284	-Ø.Ø32%	LA internacional
A# LA#	466.164	268	-Ø.Ø55%	
B SI	493.883	253	-Ø.Ø38%	

NOTA	FRECUENCIA	PERIODO	ERROR RELATIVO	
C DO	523.251	239	+Ø.Ø46%	
C# DO#	554.365	225	-Ø.215%	
D RE	587.33Ø	213	+Ø.Ø81%	
D# RE#	622.254	2Ø1	+Ø.Ø58%	
E MI	659.255	19Ø	+Ø.2Ø6%	
F FA	698.457	179	+Ø.Ø19%	
F# FA#	739.989	169	+Ø.Ø46%	Octava 1
G SOL	783.991	159	-Ø.277%	
G# SOL#	83Ø.6Ø9	15Ø	-Ø.328%	
A LA	88Ø.ØØØ	142	-Ø.Ø32%	
A# LA#	932.328	134	-Ø.Ø55%	
B SI	987.767	127	+Ø.356%	

NOTA	FRECUENCIA	PERIODO	ERROR RELATIVO	
C DO	1Ø46.5Ø2	119	-Ø.374%	
C# DO#	11Ø8.731	113	+Ø.229%	
D RE	1174.659	1Ø6	-Ø.39Ø%	
D# RE#	1244.5Ø8	1ØØ	-Ø.441%	
E MI	1318.51Ø	95	+Ø.2Ø6%	
F FA	1396.913	89	-Ø.543%	
F# FA#	1479.978	84	-Ø.548%	Octava 2
G SOL	1567.982	8Ø	+Ø.35Ø%	
G# SOL#	1661.219	75	-Ø.328%	
A LA	176Ø.ØØØ	71	-Ø.Ø32%	
A# LA#	1864.655	67	-Ø.Ø55%	
B SI	1975.533	63	-Ø.435%	

NOTA	FRECUENCIA	PERIODO	ERROR RELATIVO	
C DO	2Ø93.ØØ4	6Ø	+Ø.462%	
C# DO#	2217.461	56	-Ø.662%	
D RE	2349.318	53	-Ø.39Ø%	
D# RE#	2489.Ø16	5Ø	-Ø.441%	
E MI	2637.Ø21	47	-Ø.855%	
F FA	2793.826	45	+Ø.574%	
F# FA#	2959.955	42	-Ø.548%	Octava 3
G SOL	3135.963	4Ø	+Ø.45Ø%	
G# SOL#	3322.438	38	+Ø.992%	
A LA	352Ø.ØØØ	36	+1.357%	
A# LA#	3729.31Ø	34	+1.417%	
B SI	3951.Ø66	32	+1.134%	

NOTA	FRECUENCIA	PERIODO	ERROR RELATIVO	
C DO	4186.ØØ9	3Ø	+Ø.462%	
C# DO#	4434.922	28	-Ø.662%	
D RE	4698.636	27	+1.469%	
D# RE#	4978.Ø32	25	-Ø.441%	
E MI	5274.Ø41	24	+1.246%	
F FA	5587.652	22	-1.685%	
F# FA#	5919.911	21	-Ø.548%	Octava 4
G SOL	6271.927	2Ø	+Ø.35Ø%	
G# SOL#	6644.875	19	+Ø.992%	
A LA	7Ø4Ø.ØØØ	18	+1.357%	
A# LA#	7458.621	17	+1.417%	
B SI	79Ø2.133	16	+1.134%	


Tabla de notas, frecuencias y períodos de tono para AMSTRAD e IBM.

Por último, los MSX también cuentan con la instrucción SOUND para la generación de sonidos. En este caso el formato es:

SOUND R,CS

siendo el parámetro R el número de registro del chip de sonido sobre el que se va a actuar, y CS la característica sonora que aporta dicho registro, pudiendo ser cualquier valor comprendido entre 0 y 255.

Podemos actuar sobre 14 registros distintos numerados del 0 al 13.

Los MSX disponen de tres canales de sonido. El tono de cada canal se determina cargando los valores de frecuencia deseados en los registros del 0 al 5 (0-1 canal A, 2-3 canal B, 4-5 canal C).

El registro 6 permite la generación de ruido.

El registro 7 sirve para seleccionar los canales que emiten tono y los que emiten ruido.

Los registros 8, 9 y 10 permiten ajustar el volumen de los canales A, B y C, respectivamente.

Los registros 11 y 12 permiten controlar la envolvente de la onda sonora.

Finalmente, el registro 13 define el tipo de envolvente a utilizar.

En la figura 3 podemos ver una tabla esquemática de estos 14 registros.

N.º Registro	Función	Bit							
		b7	b6	b5	b4	b3	b2	b1	b0
0	Frecuencia canal A	FT (A)							
1						CT (A)			
2	Frecuencia canal B	FT (B)							
3						CT (B)			
4	Frecuencia canal C	FT (C)							
5						CT (C)			
6	Frecuencia ruido				NP				
7	Selección canal salida			Ruido			Tono		
		I	O	C	B	A	C	B	A
8	Sonoridad canal A				M	L (A)			
9	Sonoridad canal B				M	L (B)			
10	Sonoridad canal C				M	L (C)			
11	Periodo de la envolvente	FT (E)							
12		CT (E)							
13	Pauta de la envolvente					EP			



Tabla de funciones de los registros de sonido en MSX.

El programa 6 es una pequeña muestra del funcionamiento de SOUND en los MSX. El efecto producido es el de un helicóptero en vuelo.

```
10 REM *****************
20 REM *  HELICOPTERO   *
30 REM *      MSX       *
40 REM *****************
50 CLS
60 LOCATE 15,11:PRINT "HELICOPTERO"
70 FOR I=0 TO 13
80 READ D
90 SOUND I,D
100 NEXT I
110 RESTORE
120 GOTO 70
130 DATA 200,15,200,15,200,15
140 DATA 0,7,23,23,23,255,1,12
```



El ordenador-piano

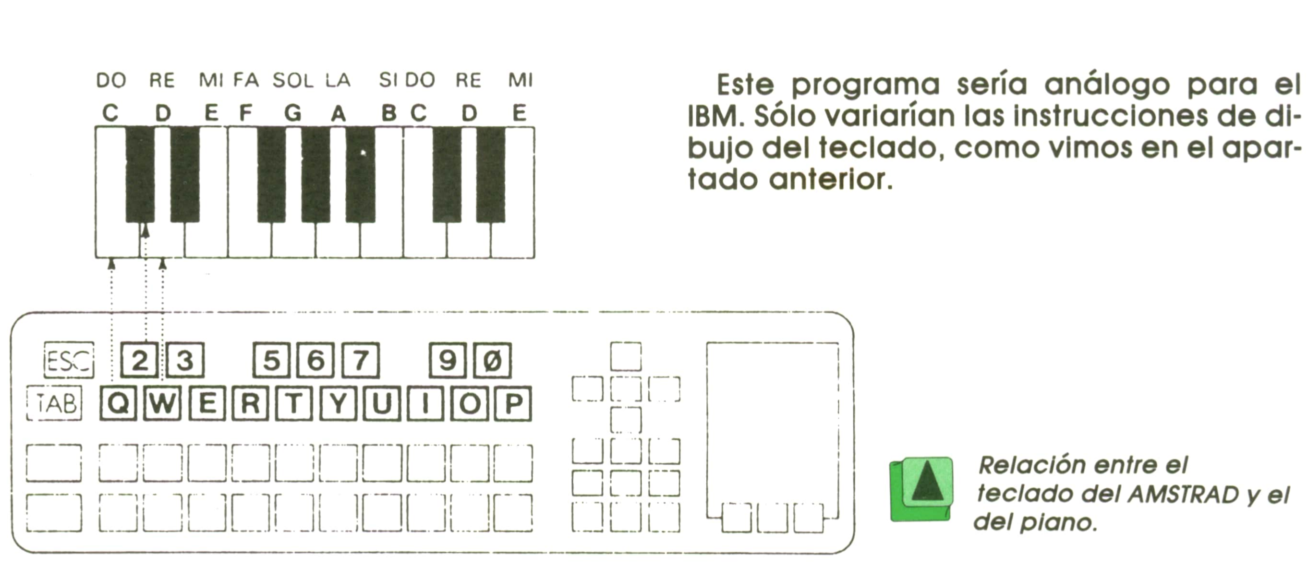
Para terminar, el programa 7 transforma el teclado de un AMSTRAD en un divertido piano. En pantalla aparece el dibujo de las teclas del piano junto con la tecla del ordenador correspondiente a cada una, siguiendo la relación mostrada en la figura 4.

El trazado del piano en la pantalla se realiza con la subrutina de la línea 220.

En la línea 70 almacenamos en la variable T$ una cadena con todas las teclas seleccionadas para simular el piano. A continuación un bucle WHILE-WEND nos permite tocar tantas notas como deseemos, ya que es infinito. Para que se produzca el sonido de una nota cada vez que pulsamos una tecla utilizamos la función INKEY$. La función INSTR de la línea 160 determina la posición que ocupa la tecla pulsada en la cadena T$. Dicho valor permite calcular la frecuencia de la nota (línea 170) y a partir de ella el tono (línea 180). La línea 190 es la que se encarga finalmente de producir el sonido correspondiente.

```
10 REM **********************
20 REM *  TECLADO DE PIANO  *
30 REM *       AMSTRAD      *
40 REM **********************
50 CLS
60 GOSUB 180
70 T$="Q2W3ER5T6Y7UI9O0P@[AZSXCFVGBNJMK,L.Ñ"
80 WHILE W=0
90 LET N$=INKEY$:IF N$="" THEN GOTO 90
100 IF N$="1" OR N$="4" OR N$="8" THEN GOTO 90
110 IF N$="-" OR N$="D" OR N$="H" THEN GOTO 90
120 IF N$=":" OR N$=";" OR N$="]" OR N$="\" THEN GOTO 90
130 LET A= ASC(N$)
140 IF A=9 OR A=16 OR A=127 THEN GOTO 90
150 IF A=13 OR A=32 OR A=224 THEN GOTO 90
160 LET N=INSTR(T$,N$)
```

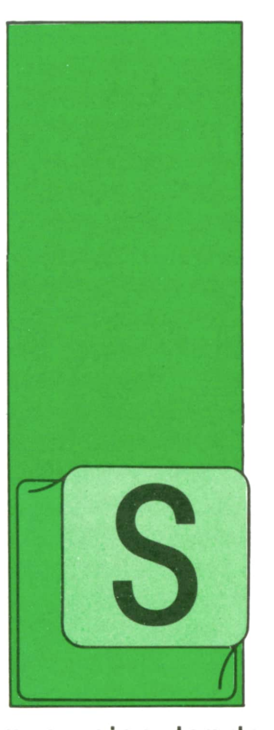
```
170 LET F=440*(2^(0+(N-10)/12))
180 LET T=ROUND(125000!/F)
190 SOUND 1,T,30,15
200 WEND
210 END
220 REM *  DIBUJO DEL TECLADO  *
230 FOR I=10 TO 30 STEP 2
240 FOR J=4 TO 6
250 IF I=14 OR I=22 OR I=28 THEN GOTO 270
260 LOCATE I,J:PRINT CHR$(143)
270 NEXT J:NEXT I
280 MOVE 120,256
290 DRAWR 384,0
300 MOVE 120,350
310 DRAWR 384,0
320 FOR X=120 TO 504 STEP 32
330 MOVE X,256
340 DRAWR 0,94
350 NEXT X
360 FOR I=10 TO 26 STEP 2
370 FOR J=14 TO 16
380 IF I=14 OR I=20 THEN GOTO 400
390 LOCATE I,J:PRINT CHR$(143)
400 NEXT J:NEXT I
410 MOVE 152,96
420 DRAWR 320,0
430 MOVE 152,190
440 DRAWR 320,0
450 FOR X=152 TO 472 STEP 32
460 MOVE X,96
470 DRAWR 0,94
480 NEXT X
490 LOCATE 10,13:PRINT "2 3   5 6 7   9 0   ^"
500 LOCATE 9,10:PRINT "Q W E R T Y U I O P @ ["
510 LOCATE 10,13:PRINT "A S   F G   J K L"
520 LOCATE 11,20:"Z X C V B N M , . /"
530 RETURN
```

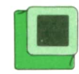


Relación entre el teclado del AMSTRAD y el del piano.

Este programa sería análogo para el IBM. Sólo variarían las instrucciones de dibujo del teclado, como vimos en el apartado anterior.

MAQUINA 8088





Instrucciones de transferencia

Se denominan así a las instrucciones que sirven para hacer que el microprocesador altere el orden normal de ejecución, que es el orden secuencial, para continuar ejecutando instrucciones contenidas en otra zona de la memoria.

El microprocesador 8088 funciona ejecutando en todo momento la instrucción apuntada por la pareja de registros CS:IP. Cuando finaliza una instrucción que no es de transferencia, la unidad de ejecución se encarga de preparar el registro IP para que apunte a la instrucción siguiente. Esto lo consigue sumando a IP la longitud (número de bytes) de la instrucción que acaba de ejecutar. Sin embargo, en las instrucciones de transferencia, el registro IP, y en ocasiones también el CS, se cargan con valores que apuntan a otra posición de memoria, provocando con esto que se transfiera la ejecución a dicha posición.

Hay cuatro clases de instrucciones de transferencia:

— Incondicionales.
— Condicionales.
— Iteraciones.
— Interrupciones.

Las instrucciones de transferencia incondicionales, como su nombre indica, provocan la transferencia sin depender de ninguna condición. Pertenecen a este grupo las instrucciones CALL, RET y JMP. Las dos primeras se describieron en el tomo 19 y la tercera se explica a continuación.

Hay 18 instrucciones de transferencia condicional (que se verán más adelante), que provocan la transferencia o no, dependiendo del estado de los «flags» del microprocesador.

Las instrucciones de iteración son: LOOP, LOOPZ (o LOOPE), LOOPNZ (o LOOPNE) y JCXZ. Estas instrucciones efectúan la transferencia dependiendo también de una condición, que es el contenido del registro CX. Dicho registro es utilizado por el microprocesador como contador interno de bucles.

El último grupo lo forman las instrucciones INT e IRET, ya estudiadas en el tomo 25, que junto con la INTO forman el grupo de instrucciones de interrupción y que actúan realizando transferencias incondicionales.

El nemotécnico INTO significa «interrupción en caso de overflow». «Overflow» es un término informático que no suele traducirse al español y significa que, en una operación aritmética, se ha perdido el dígito binario más significativo de un número, debido a que se ha excedido la capacidad del registro que lo contiene. El «flag» OF se pone en «on» cada vez que se produce esta situación.

La instrucción INTO comparte las principales características de las interrupciones llamadas con INT, pero se diferencia en dos aspectos:

— En su sintaxis, porque en vez de distinguirse de las demás por medio de un operando numérico, se distingue por la adición de la letra O al nemotécnico.
— Y en la ejecución, que está condicionada a que esté en «on» el «flag» de «overflow» (OF).



La instrucción JMP

Esta es una de las sentencias básicas del ensamblador, con la cual se puede transferir el control de ejecución a cualquier otra parte de un programa, de forma incondicional.

Se le suele denominar también instrucción de «salto» o «bifurcación» y es aná-

loga a la instrucción CALL en cuanto a sus modalidades, su sintaxis y la forma de calcular las direcciones donde realizar las transferencias, diferenciándose únicamente en el hecho de no guardar en la pila la dirección de retorno.

El formato de la sentencia JMP es el siguiente:

```
[etiqueta:]   JMP     objetivo
```

Etiqueta. Es un nombre opcional que, como en todas las sentencias ejecutables, puede servir para referenciar a esta instrucción.

JMP. Es el nemotécnico de la instrucción, abreviatura de la palabra inglesa «jump», que significa «saltar».

Objetivo. Es el operando mediante el cual se especifica la dirección a la que debe transferirse la ejecución. El objetivo puede ser: una etiqueta, un registro (que no sea un registro de segmento) o una referencia explícita a la memoria (ver tomo número 13).

Igual que las CALL, las instrucciones JMP pueden clasificarse en «intra-segmento» e «inter-segmentos», según que la dirección de memoria objetivo se referencie con el mismo registro de segmento o con un registro de segmento diferente del que tiene la propia instrucción JMP. Por otra parte, también pueden clasificarse en «directas» e «indirectas», dependiendo de si el objetivo especifica directamente la dirección de bifurcación o define una posición de memoria, que es la que contiene la dirección de bifurcación. En este caso, la citada posición de memoria puede definirse empleando todas las formas descritas en el apartado «Tipos de direccionamiento de la memoria», del tomo 13.

A continuación se dan algunos ejemplos de instrucciones JMP, así como de la forma de definir sus operandos.

```
ETIQUETA-N:                             ; Etiqueta tipo NEAR
ETIQUETA-N    LABEL NEAR                ; Etiqueta tipo NEAR
ETIQUETA-N    PROC  NEAR                ; Etiqueta tipo NEAR
                                        ; ---------
ETIQUETA-F    LABEL FAR                 ; Etiqueta tipo FAR
ETIQUETA-F    PROC  FAR                 ; Etiqueta tipo FAR
                                        ; ---------
MEMORIA-1     DB        0               ; Posición de memoria de 1 byte.
MEMORIA-2     DW        0               ; Posición de memoria de 1 palabra.
MEMORIA-4     DD        0               ; Posición de memoria de 2 palabras.
;=======================================================================
                                        ; intra-segmento directa
         JMP  ETIQUETA-N                ; ejemplo 1
                                        ; ---------
                                        ; intra-segmento indirectas
         JMP  AX                        ; ejemplo 2
         JMP  WORD PTR MEMORIA-1        ; ejemplo 3
         JMP  MEMORIA-2                 ; ejemplo 4
         JMP  WORD PTR MEMORIA-4        ; ejemplo 5
         JMP  WORD PTR [BX]             ; ejemplo 6
                                        ; ---------
                                        ; inter-segmentos directas
         JMP  ETIQUETA-F                ; ejemplo 7
         JMP  FAR PTR ETIQUETA-N        ; ejemplo 8
                                        ; ---------
                                        ; inter-segmentos indirectas
         JMP  MEMORIA-4                 ; ejemplo 9
         JMP  DWORD PTR MEMORIA-1       ; ejemplo 10
         JMP  DWORD PTR MEMORIA-2       ; ejemplo 11
         JMP  DWORD PTR [BP]            ; ejemplo 12
```

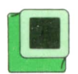
— En las instrucciones JMP intra-segmento directas, el objetivo es una etiqueta tipo NEAR (ejemplo 1), es decir, una etiqueta perteneciente al propio segmento.

— En las instrucciones JMP intra-segmento indirectas, el objetivo es un registro que no sea registro de segmento (ejemplo 2), o una posición de memoria (ejemplos 3, 4, 5 y 6). En ambos casos debe cargarse previamente la dirección de memoria a la que se quiere bifurcar.

— En las instrucciones JMP inter-segmentos directas, el objetivo es una etiqueta que se ha declarado como etiqueta tipo FAR (ejemplo 7), o una etiqueta a la que interesa, por alguna razón, efectuar una transferencia inter-segmentos, a pe-

sar de estar declarada como tipo NEAR. En este caso hay que anteponer al nombre de la etiqueta las palabras FAR PTR (ejemplo 8).

— En las instrucciones JMP inter-segmentos indirectas, el objetivo es una posición de memoria que contiene dos palabras, la primera es el desplazamiento y la segunda el segmento, que definen la dirección a la que se quiere bifurcar. En este caso, la citada posición de memoria debe haberse definido como DWORD (ejemplo 9), o ir precedida de las palabras DWORD PTR (ejemplos 10, 11 y 12). Por supuesto, debe cargarse previamente en dicha posición la dirección de memoria a la que se quiere bifurcar.



Transferencias condicionales

El 8088 dispone de 18 instrucciones que efectúan transferencias condicionadas al estado de los flags. Es decir, son instrucciones parecidas al JMP que sólo actúan si determinadas condiciones de los «flags» propias de cada instrucción resultan verdaderas. En caso contrario la ejecución continúa con la instrucción siguiente.

El formato de estas instrucciones es el siguiente:

```
[etiqueta:]    JXXX    objetivo
```

Etiqueta. Es un nombre opcional.

JXXX. Representa 30 nemotécnicos de las 18 instrucciones. Dichos nemotécnicos se forman con la letra J (inicial de jump), seguida opcionalmente de la letra N (inicial de not), y a continuación una letra que puede ser: E (inicial de «equal», que significa igual), L («less», menor), G («greater», mayor), A («above», arriba), B («below», abajo), P («parity», paridad), C («carry», acarreo), O («overflow»), S («sign», signo).

Resultan así 18 instrucciones de salto condicional asociadas a 9 condiciones afirmativas (JE, JL, JG, JA, JB, JP, JC, JO y JS), y 9 negativas (JNE, JNL, JNG, JNA, JNB, JNP, JNC, JNO y JNS). Los 12 nemotécnicos restantes son equivalentes a algunos de los 18 reseñados. Por ejemplo: la instrucción JNL (bifurcar si no menor) se puede expresar también como JGE (bifurcar si mayor o igual).

Objetivo. Es el operando mediante el cual se especifica la dirección a la que debe transferirse la ejecución. El objetivo de estas instrucciones tiene que ser forzosamente una etiqueta tipo NEAR que corresponda a una posición de memoria que no esté situada a una distancia superior a 127 bytes. A estas etiquetas cercanas se les denomina tipo SHORT.

A continuación se da una tabla de todos los nemotécnicos válidos de instrucciones de transferencias condicionales, junto con la condición necesaria para que se realicen dichas transferencias. La tabla está dividida en dos mitades de nueve instrucciones, unas inversas de las otras. También se reflejan en la tabla los nemotécnicos equivalentes de cada instrucción.

Estas instrucciones van siempre precedidas por instrucciones que establecen los estados de los «flags», como son las instrucciones aritméticas (CMP, ADD, SUB, etcétera), lógicas (NOT, AND, OR, XOR, TEST), o específicas de manipulación de «flags» (STI, CLI, STC, CLC, etc.), que explicaremos en el próximo tomo, número 40.

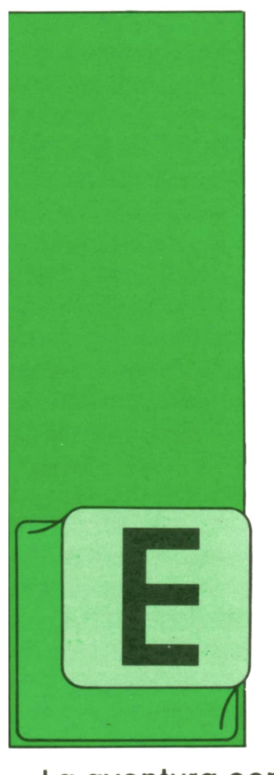
Instrucción				Instrucción inversa			
nemotécnico		nemotécnico equivalente		nemotécnico		nemotécnico equivalente	
	condición		condición		condición		condición
JE	igual	JZ	cero	JNE	no igual	JNZ	no cero
JL	menor	JNGE	no mayor/igual	JNL	no menor	JGE	mayor/igual
JG	mayor	JNLE	no menor/igual	JNG	no mayor	JLE	menor/igual
JA	arriba	JNBE	no abajo/igual	JNA	no arriba	JBE	abajo/igual
JB	abajo	JNAE	no arriba/igual	JNB	no abajo	JAE	arriba/igual
JP	paridad	JPE	paridad par	JNP	no paridad	JPO	paridad impar
JC	acarreo			JNC	no acarreo		
JO	overflow			JNO	no overfl.		
JS	signo			JNS	no signo		

PROGRAMAS

EDUCATIVOS • DE UTILIDAD • DE GESTION • DE JUEGOS



Star trek para IBM



STE programa, que ya apareció hace unos tomos en la versión del SPECTRUM, es una nueva versión de uno de los primeros juegos que aparecieron para ordenador.

La aventura consiste en destruir una serie de naves invasoras que han entrado en nuestra galaxia y que quieren hacerse con el poder de la Federación.

El jugador es el capitán de la nave Aventura, que está provista de los últimos adelantos técnicos. Podemos usar los siguientes elementos.

— Pantalla de corto alcance. Para ver nuestro sector actual.

— Pantalla de largo alcance. Para ver nueve sectores de una vez.

— Pantalla con el mapa acumulado de la galaxia. Nos indica qué hay en cada uno de los sectores de la galaxia por los que ya hemos pasado.

— Canon de rayos láser.

— Canon de torpedos.

También disponemos de un ordenador que nos permitirá calcular las trayectorias que debemos de seguir para acercarnos a nuestras bases para reabastecernos de energía y torpedos, así como para sacar la trayectoria que han de tener los misiles para dar a los enemigos.

Como los alienígenas también nos disparan, tenemos un equipo técnico que se encargará de arreglar la nave y de controlar los daños, pero en último extremo todo será de nuestra responsabilidad.

```
PROGRAMA: STAR TREK
===================

1000 REM ***************************
1010 REM *                         *
1020 REM *   S T A R  -  T R E K   *
1030 REM *                         *
1040 REM ***************************
1050 REM
1060 REM ********************************
1070 REM *                              *
1080 REM * PROGRAMADO POR:              *
1090 REM *                              *
1100 REM *       Manuel Alfonseca       *
1110 REM *              y               *
1120 REM *      Francisco  morales      *
1130 REM *                              *
1140 REM ********************************
1150 REM
1160 REM ********************************
1170 REM *                              *
1180 REM * (c) Ediciones Siglo Cultural *
1190 REM *                              *
1200 REM * (c) 1987                     *
1210 REM *                              *
```

```
1220 REM ********************************
1230 REM
1240 REM ************************
1250 REM * DEFINICION DE TABLAS *
1260 REM ************************
1270 REM
1280 DIM G(8,8)     : REM MAPA DE LA GALAXIA
1290 DIM QQ$(8,32)  : REM MAPA DE UN CUADRANTE DE LA GALAXIA
1300 DIM ZZ(10,10)  : REM MAPA ACUMULADO DE LA GALAXIA
1310 DIM D(8)       : REM DANOS EN LA NAVE ADVENTURE
1320 DIM K(10,3)    : REM NAVES ENEMIGAS EN ESTE CUADRANTE
1330 DIM B(1,2)     : REM BASES ESTELARES EN ESTE CUADRANTE
1340 DIM E(8,2)     : REM ESTRELLAS EN ESTE CUADRANTE
1350 REM
1360 REM ******************************************
1370 REM * VARIABLES 'C' UTILIZADAS POS EL TORPEDO *
1380 REM ******************************************
1390 REM
1400 DIM CA(8),CB(8),CC(8),CD(8)
1410 REM
1420 REM *** VARIABLES 'CA' ***
1430 REM
1440 LET CA(1)=0
1450 LET CA(2)=-1
1460 LET CA(3)=-1
1470 LET CA(4)=-1
1480 LET CA(5)=0
1490 LET CA(6)=1
1500 LET CA(7)=1
1510 LET CA(8)=1
1520 REM
1530 REM *** VARIABLES 'CB' ***
1540 REM
1550 LET CB(1)=-1
1560 LET CB(2)=0
1570 LET CB(3)=0
1580 LET CB(4)=1
1590 LET CB(5)=1
1600 LET CB(6)=0
1610 LET CB(7)=0
1620 LET CB(8)=-1
1630 REM
1640 REM *** VARIABLES 'CC' ***
1650 REM
1660 LET CC(1)=1
1670 LET CC(2)=1
1680 LET CC(3)=0
1690 LET CC(4)=-1
1700 LET CC(5)=-1
1710 LET CC(6)=-1
1720 LET CC(7)=0
1730 LET CC(8)=1
1740 REM
1750 REM *** VARIABLES 'CD' ***
1760 REM
1770 LET CD(1)=0
1780 LET CD(2)=-1
1790 LET CD(3)=-1
1800 LET CD(4)=0
1810 LET CD(5)=0
1820 LET CD(6)=1
1830 LET CD(7)=1
1840 LET CD(8)=0
1850 REM
1860 REM ************************************
1870 REM * CREACION DE LA AVENTURA ESPACIAL *
1880 REM ************************************
1890 REM
```

```
1900 KEY OFF: FOR I=1 TO 10: KEY I,"": NEXT I
1910 RANDOMIZE TIMER
1920 LET B9=0:LET K9=0
1930 REM
1940 REM ****************************************************
1950 REM * GENERACION DE ESTRELLAS, BASES Y NAVES ENEMIGAS *
1960 REM ****************************************************
1970 REM
1980 FOR I=1 TO 8
1990    FOR J=1 TO 8
2000       LET A=RND(1)
2010       LET G(I,J)=AS+10*AB+100*AK
2020       IF A>.8 THEN LET AK=1
2030       IF A>.95 THEN LET AK=2
2040       IF A>.9799999 THEN LET AK=3
2050       LET K9=K9+AK
2060       LET A=RND(1)
2070       LET AB=0
2080       IF A>.96 THEN LET AB=1
2090       LET B9=B9+AB
2100       LET AS=1+INT(8*RND)
2110       LET G(I,J)=AS+10*AB+100*AK
2120    NEXT J
2130 NEXT I
2140 LET K7=K9
2150 REM
2160 REM *** POSICION DE LA NAVE "AVENTURA" ***
2170 REM
2180 LET Q1=1+INT(8*RND):LET Q2=1+INT(8*RND)
2190 LET S1=1+INT(8*RND):LET S2=1+INT(8*RND)
2200 REM
2210 REM *** CALCULO DE FECHAS ***
2220 REM
2230 LET TO=100*(20+INT(20*RND))         : REM FECHA INICIAL
2240 LET T=TO                            : REM FECHA ACTUAL
2250 LET T9=30: IF K9>20 THEN LET T9=35  : REM FECHA FINAL
2260 REM
2270 REM *** OTROS DATOS ***
2280 REM
2290 LET EO=3000 : REM ENERGIA INICIAL
2300 LET E1=EO   : REM ENERGIA ACTUAL
2310 LET PO=10   : REM NUMERO INICIAL DE TORPEDOS
2320 LET P1=PO   : REM NUMERO ACTUAL DE TORPEDOS
2330 LET SO=0    : REM ESCUDO PROTECTOR ACTUAL
2340 REM
2350 REM ************************
2360 REM * COMIENZA LA AVENTURA *
2370 REM ************************
2380 REM
2390 CLS
2400 LOCATE 1,32:PRINT "S T A R    T R E K"
2410 PRINT TAB(31);"==================="
2420 PRINT
2430 PRINT "AL CAPITAN DE LA NAVE AVENTURA. ESTAS SON LAS ORDENES:"
2440 PRINT
2450 PRINT " Debe usted destruir las";K9;"naves enemigas que han invadido la"
2460 PRINT "galaxia, antes de que puedan atacar el cuartel general de la fede-"
2470 PRINT "racion."
2480 PRINT
2490 PRINT " El ultimo dia habil es el";TO+T9;".Esto le deja";T9;"fechas"
2500 PRINT "estelares para completar la mision."
2510 PRINT
2520 PRINT " Hay";B9;"bases estelares en la galaxia. En ellas te puedes reabas-"
2530 PRINT "tecer de gasolina y de torpedos."
2540 PRINT
2550 PRINT "    BUENA SUERTE ...... (LA VA A NECESITAR)"
2560 PRINT:PRINT
2570 PRINT "PULSE UNA TECLA CUANDO ESTE LISTO PARA COMENZAR LA AVENTURA"
```

```
2580 A$=INKEY$
2590 IF A$="" THEN GOTO 2580
2600 REM
2610 REM *** INICIALIZAR PANTALLA ***
2620 REM
2630 CLS
2640 LOCATE 2,6:COLOR 1,7
2650 PRINT "     PANTALLA DE CORTO ALCANCE      ":COLOR 7,0
2660 LOCATE 3,44
2670 PRINT "FECHA ESTELAR"
2680 LOCATE 4,44
2690 PRINT "CONDICION"
2700 LOCATE 5,44
2710 PRINT "CUADRANTE"
2720 LOCATE 6,44
2730 PRINT "SECTOR"
2740 LOCATE 7,44
2750 PRINT "ENERGIA TOTAL"
2760 LOCATE 8,44
2770 PRINT "TORPEDOS"
2780 LOCATE 9,44
2790 PRINT "ESCUDO PROTECTOR"
2800 REM
2810 REM *** SALA DE CONTROL ***
2820 REM
2830 GOSUB 2840:GOTO 3080
2840 LOCATE 13,6:PRINT "PANTALLA DE LARGO ALCANCE":LOCATE 14,6
2850 IF D(3)<0 THEN GOTO 2990
2860 PRINT "-----------"
2870 FOR I=1 TO 3
2880    LOCATE 14+I,6
2890    FOR J=1 TO 3
2900       LET A=0
2910       LET A1=Q1+I:LET A2=Q2+J
2920       IF (A1>2) AND (A1<11) AND (A2>2) AND (A2<11) THEN LET A=G(A1-2,A2-2)
2930       LET ZZ(Q1+I-1,Q2+J-1)=A
2940       PRINT USING "###";A;
2950    NEXT J
2960 NEXT I
2970 LOCATE 18,6: PRINT "-----------"
2980 RETURN
2990 GOSUB 7930
3000 PRINT "PANTALLA DE LARGO ALCANCE ESTROPEADA"
3010 FOR I=1 TO 5
3020    LOCATE 13+I,6
3030    FOR J=1 TO 3
3040       PRINT "       " : REM 7 ESPACIOS
3050    NEXT J
3060 NEXT I
3070 RETURN
3080 REM
3090 REM *** ENTRAMOS EN UN NUEVO CUADRANTE ***
3100 REM
3110 LOCATE 5,60
3120 PRINT Q1;",";Q2
3130 LET S3=G(Q1,Q2):LET K3=INT(S3/100):LET S3=S3-100*K3:LET B3=INT(S3/10):LET S
3=S3-10*B3
3140 REM
3150 REM *** DISTRIBUCION DE OBJETOS EN EL CUADRANTE ***
3160 REM
3170 FOR I=1 TO 8
3180    FOR J=1 TO 32
3190       LET QQ$(I,J)=" "
3200    NEXT J
3210 NEXT I
3220 LET QQ$(S1,1+4*(S2-1))="<"
3230 LET QQ$(S1,2+4*(S2-1))="="
3240 LET QQ$(S1,3+4*(S2-1))=">"
```

```
3250 LET F1$="+":LET  F2$="+":LET F3$="+"
3260 FOR I=1 TO K3
3270    GOSUB 3490
3280    LET K(I,1)=R1
3290    LET K(I,2)=R2
3300    LET K(I,3)=200
3310 NEXT I
3320 IF K3=3 THEN GOTO 3360
3330 FOR I=K3+1 TO 3
3340    LET K(I,3)=0
3350 NEXT I
3360 LET F1$="-":LET  F2$="O":LET F3$="-"
3370 FOR I=1 TO B3
3380    GOSUB 3490
3390    LET B(I,1)=R1
3400    LET B(I,2)=R2
3410 NEXT I
3420 LET F1$=" ":LET F2$="*":LET F3$=" "
3430 FOR I=1 TO S3
3440    GOSUB 3490
3450    LET E(I,1)=R1
3460    LET E(I,2)=R2
3470 NEXT I
3480 GOTO 3550
3490 LET R1=1+INT(8*RND(1)):LET R2=1+INT(8*RND(1))
3500 IF QQ$(R1,2+4*(R2-1))<>" " THEN GOTO 3490
3510 LET QQ$(R1,1+4*(R2-1))=F1$
3520 LET QQ$(R1,2+4*(R2-1))=F2$
3530 LET QQ$(R1,3+4*(R2-1))=F3$
3540 RETURN
3550 REM
3560 REM *** COMPROBACION DE POSICION DE AMARRE CON ESTACION ***
3570 REM
3580 IF B3=0 THEN GOTO 3660          : REM NO HAY BASE
3590 IF S2<>B(1,2) THEN GOTO 3660    : REM NO ESTAMOS AMARRADOS
3600 IF (S1<B(1,1)-1) OR (S1>B(1,1)+1) THEN GOTO 3660
3610 LOCATE 4,60: PRINT "EN PUERTO       "
3620 LET D0=1                        : REM EN PUERTO
3630 LET E1=E0: LET P1=P0: LET S0=0 : REM REPOSTAMOS
3640 GOSUB 7930: PRINT "SIN ESCUDO PROTECTOR PARA EL AMARRE"
3650 GOTO 3770
3660 LET D0=0                        : REM NO ESTAMOS AMARRADOS
3670 IF K3>0 THEN GOTO 3710
3680 IF E1<.1*E0 THEN GOTO 3760
3690 LOCATE 4,60: PRINT "NORMAL (VERDE) "
3700 GOTO 3770
3710 LOCATE 4,60: COLOR 2: PRINT "ALERTA ROJA    ":COLOR 7
3720 PLAY "O3T150L8MSAAAP8AAA"
3730 IF S0>200 THEN GOTO 3770
3740 GOSUB 7930: PRINT "ESCUDO PROTECTOR PELIGROSAMENTE BAJO"
3750 GOTO 3770
3760 LOCATE 4,60: COLOR 14: PRINT "ALERTA AMARILLA": COLOR 7
3770 IF D(2)<0 THEN GOTO 3860        : REM PANTALLA DE CORTO ALCANCE
3780 LOCATE 3,6:PRINT "--------------------------------"
3790 FOR I=1 TO 8
3800    LOCATE 3+I,6
3810    FOR J=1 TO 32
3820       PRINT QQ$(I,J);
3830    NEXT J
3840 NEXT I
3850 GOTO 3930
3860 GOSUB 7930: PRINT "PANTALLA DE CORTO ALCANCE ESTROPEADA"
3870 FOR I=1 TO 8
3880    LOCATE 3+I,6
3890    FOR J=1 TO 32
3900       PRINT " ";
3910    NEXT J
3920 NEXT I
```

```
3930 LOCATE 12,6:PRINT "--------------------------------"
3940 LOCATE 3,60: PRINT T;"     "
3950 LOCATE 6,60: PRINT S1;",";S2;"     "
3960 LOCATE 7,60: PRINT E1+SO;"     "
3970 LOCATE 8,60: PRINT P1;"     "
3980 LOCATE 9,60: PRINT SO;"     "
3990 REM
4000 REM *** COMPROBACION DE ENERGIA SUFICIENTE ***
4010 REM
4020 IF SO+E1<10 THEN GOTO 4040
4030 IF (E1>10) OR (D(7)=0) THEN GOTO 4130
4040 PLAY "O3T150L8MSAAAP8AAA"
4050 CLS
4060 PRINT "--- ERROR FATAL!!! LA NAVE HA QUEDADO VARADA EN EL ESPACIO"
4070 PRINT "NO HAY ENERGIA SUFICIENTE, Y EL CONTROL DE ESCUDO NO PUEDE"
4080 PRINT "PASAR ENERGIA A LA SALA DE MAQUINAS."
4090 GOTO 8090
4100 REM
4110 REM *** BUCLE DE PETICION DE ORDENES ***
4120 REM
4130 LOCATE 22,44: PRINT "ORDENES:             "
4140 LOCATE 23,44: PRINT "--------             "
4150 LOCATE 24,1:COLOR 0,7
4160 PRINT " N=NAVEGACION  C=CANON  T=TORPEDO  E=ESCUDO  D=DANOS  O=ORDENADOR
     F=FIN ";
4170 COLOR 7,0
4180 GOSUB 7140                          : REM LEE UNA TECLA
4190 IF I=70 THEN CLS:GOTO 8100          : REM FIN DEL JUEGO
4200 IF I=78 OR I=110 THEN GOTO 4270 : REM NAVEGACION
4210 IF I=67 OR I=99  THEN GOTO 4980 : REM CANON
4220 IF I=84 OR I=116 THEN GOTO 5270 : REM TORPEDO
4230 IF I=69 OR I=101 THEN GOTO 5630 : REM ESCUDO
4240 IF I=68 OR I=100 THEN GOTO 5760 : REM DANOS
4250 IF I=79 OR I=111 THEN GOTO 6170 : REM ORDENADOR
4260 GOTO 4180
4270 REM
4280 REM *** CONTROL DE NAVEGACION ***
4290 REM
4300 LOCATE 22,44:PRINT "CURSO (1-9):          ";
4310 LET N1=1:LET N2=9:LOCATE 22,65:GOSUB 7200:IF N1=-1 THEN GOTO 4130 ELSE LET
C1=N1
4320 LOCATE 23,44:PRINT "VELOCIDAD (0-8):       ";
4330 LET N1=0:LET N2=8:LOCATE 23,65:GOSUB 7200:IF N1=-1 THEN GOTO 4130 ELSE LET
W1=N1
4340 IF C1=9 THEN LET C1=1
4350 IF (W1=<.2) OR (D(1)>=0) THEN GOTO 4390
4360 GOSUB 7930:PRINT "MOTORES DA%ADOS                   "
4370 LOCATE 15,44:PRINT "VELOCIDAD MAXIMA=0.2           "
4380 GOTO 4270
4390 LET N=INT(.5+8*W1):IF E1>=N THEN GOTO 4440
4400 GOSUB 7930:PRINT "ENERGIA INSUFICIENTE PARA MANIOBRAR"
4410 IF (D(7)<0) OR (SO<N-E1) THEN GOTO 4130
4420 LOCATE 15,44:PRINT SO;"UNIDADES DISPONIBLES EN EL ESCUDO"
4430 GOTO 4130
4440 REM
4450 REM *** COMIENZA EL MOVIMIENTO ***
4460 REM
4470 REM *** PRIMERO EL ENEMIGO ATACA ***
4480 REM
4490 GOSUB 7520
4500 REM
4510 REM *** REPARACIONES EN MARCHA ***
4520 REM
4530 GOSUB 7750
4540 LET XX1=S1:LET XX2=S2
4550 LET QQ$(INT(S1),1+4*(INT(S2)-1))=" "
4560 LET QQ$(INT(S1),2+4*(INT(S2)-1))=" "
4570 LET QQ$(INT(S1),3+4*(INT(S2)-1))=" "
```

```
4580 LET X1=CA(INT(C1))+CB(INT(C1))*(C1-INT(C1))
4590 LET X2=CC(INT(C1))+CD(INT(C1))*(C1-INT(C1))
4600 LET I=1
4610 LET S1=S1+X1:LET S2=S2+X2
4620 IF (S1<1) OR (S1>=9) OR (S2<1) OR (S2>=9) THEN GOTO 4740
4630 IF QQ$(INT(S1),2+4*(INT(S2)-1))=" " THEN GOTO 4680
4640 LET S1=S1-X1:LET S2=S2-X2
4650 GOSUB 7930:PRINT "PARADA AUTOMATICA DE MOTORES      "
4660 LOCATE 15,44:PRINT "DEBIDO A MALA NAVEGACION          "
4670 GOTO 4690
4680 LET I=I+1:IF N>=I THEN GOTO 4610
4690 LET QQ$(INT(S1),1+4*(INT(S2)-1))="<"
4700 LET QQ$(INT(S1),2+4*(INT(S2)-1))="="
4710 LET QQ$(INT(S1),3+4*(INT(S2)-1))=">"
4720 GOSUB 4900
4730 LET S1=INT(.5+S1):LET S2=INT(.5+S2):GOTO 3550
4740 LET XX1=(8*Q1)+XX1+N*X1
4750 LET XX2=(8*Q2)+XX2+N*X2
4760 LET Q1=INT(XX1/8):LET S1=INT(XX1-8*Q1)
4770 LET Q2=INT(XX2/8):LET S2=INT(XX2-8*Q2)
4780 IF S1=0 THEN LET S1=8:LET Q1=Q1-1
4790 IF S2=0 THEN LET S2=8:LET Q2=Q2-1
4800 IF Q1<1 THEN LET Q1=1:GOSUB 4860
4810 IF Q2<1 THEN LET Q2=1:GOSUB 4860
4820 IF Q1>8 THEN LET Q1=8:GOSUB 4860
4830 IF Q2>8 THEN LET Q2=8:GOSUB 4860
4840 GOSUB 4900
4850 LET S1=INT(.5+S1):LET S2=INT(.5+S2):GOTO 2800
4860 GOSUB 7930: PRINT "DETENCION AUTOMATICA DE MOTORES"
4870 LOCATE 15,44: PRINT "PERMISO DENEGADO PARA ABANDONAR"
4880 LOCATE 16,44: PRINT "LA GALAXIA                      "
4890 RETURN
4900 LET E1=E1-N-10
4910 IF E1>=0 THEN GOTO 4960
4920 GOSUB 7930:PRINT "ENERGIA TRANSFERIDA DESDE EL ESCUDO"
4930 LOCATE 15,44:PRINT "PARA COMPLETAR LA MANIOBRA"
4940 LET S0=S0+E1:IF S0<0 THEN LET S0=0
4950 LET E1=0
4960 LET TT1=.1*INT(10*W1):IF TT1>1 THEN LET TT1=1:LET T=T+TT1
4970 IF T>T0+T9 THEN GOTO 8090 ELSE RETURN
4980 REM
4990 REM *** DISPARO DE CA%ONES LASER ***
5000 REM
5010 IF K3>0 THEN GOTO 5040
5020 GOSUB 7930:PRINT "NO HAY NAVES ENEMIGAS A LA VISTA  "
5030 GOTO 4130
5040 IF D(4)>=0 THEN GOTO 5070
5050 GOSUB 7930:PRINT "EL CANON LASER NO FUNCIONA        "
5060 GOTO 4130
5070 GOSUB 7930:PRINT "CANON LASER APUNTANDO AL OBJETIVO  "
5080 LOCATE 15,44:PRINT "ENERGIA DISPONIBLE=";E1
5090 IF D(8)>=0 THEN GOTO 5110
5100 LOCATE 16,44:PRINT "MENOR PRECISION POR FALLO COMPUTADOR"
5110 LOCATE 22,44:PRINT "ENERGIA DE DISPARO:";
5120 LET N1=0:LET N2=E1:LOCATE 22,65:GOSUB 7200:IF F1=-1 THEN GOTO 4130 ELSE LET
C1=N1
5130 LET E1=E1-C1
5140 GOSUB 7520
5150 IF D(7)>=0 THEN GOTO 5170
5160 LET C1=C1*RND
5170 LET C1=INT(C1/K3)
5180 FOR I=1 TO 3
5190     IF K(I,3)=0 THEN GOTO 5250
5200     LET H=INT((2+RND)*C1/SQR(((K(I,1)-S1)*(K(I,1)-S1))+((K(I,1)-S2)*(K(I,2)-
S2))))
5210     IF H=>.15*K(I,3) THEN GOTO 5250
5220     LET K(I,3)=K(I,3)-H
5230     GOSUB 7930:PRINT H;"UNIDADES PARA ENEMIGO EN";K(I,1);K(I,2)
```

```
5240    IF K(I,3)=0 THEN LET XA=K(I,1):LET XB=K(I,2):GOSUB 7290
5250 NEXT I
5260 GOTO 3550
5270 REM
5280 REM *** LANZAMIENTO DE UN TORPEDO ***
5290 REM
5300 GOSUB 7930
5310 IF D(5)<0 THEN PRINT "LOS TUBOS DE TORPEDOS NO FUNCIONAN":GOTO 4130
5320 IF P1=<0 THEN PRINT "TORPEDOS AGOTADOS            ":GOTO 4130
5330 LOCATE 22,44:PRINT "CURSO TORPEDO (1-9): ";
5340 LET N1=1:LET N2=9:LOCATE 22,65:GOSUB 7200:IF N1=-1 THEN GOTO 4130
5350 IF N1=9 THEN LET N1=1
5360 LET X1=CA(INT(N1))+CB(INT(N1))*(N1-INT(N1))
5370 LET X2=CC(INT(N1))+CD(INT(N1))*(N1-INT(N1))
5380 LET E1=E1-2:LET P1=P1-1:LET XA=S1:LET XB=S2
5390 LOCATE 23,44:PRINT "CURSO DEL TORPEDO:"
5400 LET XA=XA+X1:LET XB=XB+X2
5410 IF (XA>=9) OR (XB>=9) OR (XA<1) OR (XB<1) THEN GOTO 5590
5420 LOCATE 23,65:PRINT XA;XB
5430 LET I$=QQ$(INT(XA+.5),2+4*INT(XB-.5))
5440 IF I$=" " THEN GOTO 5600
5450 IF I$="+" THEN GOSUB 7290:GOTO 5540
5460 IF I$="*" THEN GOTO 5580
5470 GOSUB 7930:PRINT "-BASE ESTELAR DESTRUIDA, ESTUPIDO!"
5480 LET B3=B3-1:LET B9=B9-1:LET D0=0
5490 LET QQ$(INT(XA),1+4*INT(XB-1))=" "
5500 LET QQ$(INT(XA),2+4*INT(XB-1))=" "
5510 LET QQ$(INT(XA),3+4*INT(XB-1))=" "
5520 LET G(Q1,Q2)=S3+10*(B3+10*K3)
5530 GOSUB 2840
5540 GOSUB 7520
5550 LOCATE 22,65:PRINT "       "
5560 LOCATE 23,65:PRINT "       "
5570 GOTO 3550
5580 GOSUB 7930:PRINT "UNA ESTRELLA ABSORBIO EL TORPEDO":GOTO 5540
5590 GOSUB 7930:PRINT "EL TORPEDO HA FALLADO EL OBJETIVO":GOTO 5540
5600 IF XH<>0 THEN LOCATE 3+XH,6+XK:PRINT " "
5610 LET XH=INT(XA+.5):LET XK=2+4*INT(XB-.5)
5620 LOCATE 3+XH,6+XK:PRINT "":PLAY "MFP4MB":GOTO 5400
5630 REM
5640 REM *** TRASPASO DE ENERGIA AL ESCUDO PROTECTOR ***
5650 REM
5660 GOSUB 7930
5670 IF D(7)>=0 THEN GOTO 5690
5680 GOSUB 7930:PRINT "EL ESCUDO PROTECTOR NO FUNCIONA":GOTO 4130
5690 LOCATE 22,44:PRINT "ENERGIA AL ESCUDO: ";
5700 LET N1=0:LET N2=E1+S0:LOCATE 22,65:GOSUB 7200:IF N1=-1 THEN GOTO 4130
5710 LET E1=E1+S0-N1
5720 LET S0=N1
5730 GOSUB 7930:PRINT "ESCUDO A";S0;"SEGUN SUS ORDENES"
5740 LOCATE 9,60:PRINT S0
5750 LOCATE 22,65:PRINT "         ": GOTO 4130
5760 REM
5770 REM *** INFORME DE DANOS ***
5780 REM
5790 GOSUB 7930
5800 IF D(6)<0 THEN PRINT "EL CONTROL DE DA%OS NO FUNCIONA  ":GOTO 5850
5810 FOR I=1 TO 8
5820    LOCATE 13+I,44:GOSUB 6080:PRINT DD$;D(I)
5830 NEXT I
5840 GOSUB 8020
5850 IF D0=0 THEN GOTO 6020
5860 LET D3=0
5870 FOR I=1 TO 8
5880    IF D(I)<0 THEN LET D3=D3+.1
5890 NEXT I
5900 IF D3>1 THEN LET D3=1
5910 IF D3=0 THEN GOTO 6020
```

```
5920 GOSUB 7930:PRINT "TECNICOS DISPUESTOS A REPARAR NAVE"
5930 LOCATE 15,44:PRINT "TIEMPO DE REPARACION ESTIMADO:"
5940 LOCATE 16,44:PRINT D3;"FECHAS ESTELARES"
5950 LOCATE 17,44:PRINT "(AUTORIZA USTED LAS REPARACIONES?"
5960 LOCATE 22,44:PRINT "SI/NO:     "
5970 LOCATE 22,65:INPUT I$: IF I$<>"SI" THEN GOTO 3550
5980 FOR I=1 TO 8
5990    LET D(I)=0
6000 NEXT I
6010 LET T=T+D3+.1:GOTO 5810
6020 GOSUB 2840:GOSUB 2840
6030 FOR I=1 TO 8
6040    LOCATE 13+I,44
6050    PRINT "                          "
6060 NEXT I
6070 GOTO 3550
6080 ON I GOTO 6090,6100,6110,6120,6130,6140,6150,6160
6090 DD$="MOTORES               ": RETURN
6100 DD$="PANTALLA CORTO AL.  ": RETURN
6110 DD$="PANTALLA LARGO AL.  ": RETURN
6120 DD$="CA%ONES LASER       ": RETURN
6130 DD$="TUBOS DE TORPEDOS   ": RETURN
6140 DD$="CONTROL DE DA%OS    ": RETURN
6150 DD$="CONTROL DE ESCUDO   ": RETURN
6160 DD$="COMPUTADORA         ": RETURN
6170 REM
6180 REM *** COMPUTADORA DE A BORDO ***
6190 REM
6200 GOSUB 7930
6210 IF D(8)<0 THEN GOTO 6330
6220 PRINT "COMPUTADOR ACTIVO, ESPERANDO ORDENES"
6230 LOCATE 24,1:COLOR 0,7
6240 PRINT "        G=GALAXIA  S=SITUACION  T=TORPEDOS  B=BASE  D=DIRECCION--DIST
ANCIA        ";
6250 COLOR 7,0
6260 GOSUB 7140
6270 IF I=71 OR I=103 THEN GOTO 6340
6280 IF I=83 OR I=115 THEN GOTO 6590
6290 IF I=84 OR I=116 THEN GOTO 6670
6300 IF I=66 OR I=98  THEN GOTO 6950
6310 IF I=68 OR I=100 THEN GOTO 7020
6320 GOTO 6260
6330 PRINT "LA COMPUTADORA NO FUNCIONA    ":GOTO 4130
6340 REM
6350 REM *** MAPA DE LA GALAXIA ***
6360 REM
6370 LOCATE 13,5:COLOR 1,7:PRINT "    MAPA ACUMULADO DE LA GALAXIA      ":COLOR 6
,0
6380 LOCATE 14,6:PRINT "   1   2   3   4   5   6   7   8"
6390 LOCATE 15,6:PRINT "==================================":COLOR 7,0
6400 FOR I=1 TO 8
6410    LOCATE 15+I,2
6420    PRINT I
6430    LOCATE 15+I,6
6440    FOR J=1 TO 8
6450       PRINT USING " ###";ZZ(I+1,J+1);
6460    NEXT J
6470 NEXT I
6480 LOCATE 14,44:PRINT "PRESIONE UNA TECLA PARA SEGUIR      "
6490 A$=INKEY$:IF A$="" THEN GOTO 6490
6500 LOCATE 13,5:PRINT "                                      "
6510 LOCATE 14,6:PRINT "                                 "
6520 LOCATE 15,6:PRINT "                                   "
6530 FOR I=1 TO 8
6540    LOCATE 15+I,2
6550    PRINT "                                "
6560 NEXT I
6570 LOCATE 14,44:PRINT "                                    "
```

```
6580 GOSUB 2840:GOTO 4130
6590 REM
6600 REM *** SITUACION ***
6610 REM
6620 GOSUB 7930:PRINT "INFORME DE SITUACION:"
6630 LOCATE 15,44:PRINT "QUEDAN";K9;"NAVES ENEMIGAS"
6640 LOCATE 16,44:PRINT "QUEDAN";T0+T9-T;"FECHAS ESTELARES"
6650 LOCATE 17,44:PRINT "HAY";B9;"BASES ESTELARES EN LA GALAXIA"
6660 GOTO 4130
6670 REM
6680 REM *** CALCULO DE DIRECCION PARA TORPEDOS ***
6690 REM
6700 GOSUB 7930
6710 IF K3=0 THEN PRINT "NO HAY NAVES ENEMIGAS AQUI":GOTO 4130
6720 FOR I=1 TO 3
6730    IF K(I,3)=0 THEN GOTO 6750
6740    LET N1=K(I,1):LET N2=K(I,2):GOSUB 6800
6750 NEXT I
6760 GOTO 4130
6770 REM
6780 REM *** RUTINA DE CALCULO DE DIRECCION/DISTANCIA ***
6790 REM
6800 LOCATE 14,44:PRINT " DIRECCION  DISTANCIA"
6810 LET X1=N1:LET X2=N2
6820 LET A=S1-X1:LET X=X2-S2
6830 IF X<0 THEN GOTO 6880 ELSE IF A<0 THEN GOTO 6910
6840 IF X>0 THEN GOTO 6850 ELSE IF A=0 THEN GOTO 6890
6850 LET C1=1
6860 IF ABS(A)<ABS(X) THEN LET C1=C1+2-ABS(X/A) ELSE LET C1=C1+ABS(A/X)
6870 GOTO 6930
6880 IF A>0 THEN GOTO 6900 ELSE IF X=0 THEN GOTO 6910
6890 LET C1=5:IF (A=0) AND (X=0) THEN GOTO 6930 ELSE GOTO 6860
6900 LET C1=3:GOTO 6920
6910 LET C1=7
6920 IF ABS(A)<ABS(X) THEN LET C1=C1+2-ABS(A/X) ELSE LET C1=C1+ABS(X/A)
6930 LOCATE 14+I,44:PRINT C1,SQR(X*X+A*A)
6940 RETURN
6950 REM
6960 REM *** CALCULO DE DIRECCION PARA BASES ***
6970 REM
6980 GOSUB 7930
6990 IF B3=0 THEN GOTO 7010
7000 LET N1=B(1,1):LET N2=B(1,2):LET I=1:GOSUB 6800:GOTO 4130
7010 PRINT "NO HAY BASES ESTELARES AQUI":GOTO 4130
7020 REM
7030 REM *** CALCULO DE DIRECCION CUALQUIERA ***
7040 REM
7050 LOCATE 22,44:PRINT "10 COORDENADA:";
7060 LOCATE 23,44:PRINT "20 COORDENADA:";
7070 LET N1=1:LET N2=8:LOCATE 22,65:GOSUB 7200:IF N1=-1 THEN GOTO 4130
7080 LET X1=N1:LET N1=1:LET N2=8:LOCATE 23,65:GOSUB 7200:IF N1=-1 THEN GOTO 4130
7090 LET N2=N1:LET N1=X1:IF N1=9 THEN LET N1=1
7100 IF N2=9 THEN LET N2=1
7110 LET X2=N2:LET A=Q1-X1:LET X=X2-Q2
7120 GOSUB 7930:PRINT "DIRECCION  DISTANCIA"
7130 LET I=1:GOSUB 6830:GOTO 4130
7140 REM
7150 REM *** LEE UNA TECLA DE FUNCION ***
7160 REM
7170 LET A$=INKEY$:IF A$="" THEN GOTO 7170:IF LEN(A$)<>2 THEN GOTO 7170
7180 LET I=ASC(RIGHT$(A$,1)):IF I<59 THEN GOTO 7170
7190 RETURN
7200 REM
7210 REM *** PETICION DE DATOS ***
7220 REM
7230 INPUT A
7240 IF (A<N1) OR (A>N2) THEN GOTO 7270
7250 LET N1=A:LET N2=B
```

```
7260 RETURN
7270 GOSUB 7930:PRINT "VALOR INVALIDO":LET N1=-1
7280 RETURN
7290 REM
7300 REM *** NAVE ENEMIGA DESTRUIDA ***
7310 REM
7320 PLAY "O3L8MNT120CDE2"
7330 GOSUB 7930:PRINT "-NAVE ENEMIGA DESTRUIDA!"
7340 LET K3=K3-1:LET K9=K9-1
7350 IF K9=0 THEN GOTO 7470
7360 FOR I3=1 TO 3
7370    IF XA<>K(I3,1) THEN GOTO 7390
7380    IF XB=K(I3,2) THEN GOTO 7400
7390 NEXT I3
7400 LET K(I3,3)=0
7410 LET QQ$(INT(XA+.5),1+4*INT(XB-.5))=" "
7420 LET QQ$(INT(XA+.5),2+4*INT(XB-.5))=" "
7430 LET QQ$(INT(XA+.5),3+4*INT(XB-.5))=" "
7440 LET G(Q1,Q2)=S3+10*(B3+10*K3)
7450 GOSUB 2840
7460 RETURN
7470 CLS
7480 PRINT "-ENHORABUENA CAPITAN! LA ULTIMA NAVE ENEMIGA"
7490 PRINT "HA SIDO DESTRUIDA":PRINT
7500 PRINT "SU EFICIENCIA ES IGUAL A";INT(1000*K7/(T-T0))
7510 END
7520 REM
7530 REM *** ATAQUE ENEMIGO ***
7540 REM
7550 IF K3=0 THEN RETURN
7560 IF D0=0 THEN GOTO 7590
7570 GOSUB 7930:PRINT "LA BASE PROTEGE LA NAVE AVENTURA"
7580 RETURN
7590 FOR I2=1 TO 3
7600    IF K(I2,3)=0 THEN GOTO 7720
7610    LET H=SQR(((K(I2,1)-S1)*(K(I2,1)-S1))+((K(I2,2)-S2)*(K(I2,2)-S2)))
7620    LET H=INT((2+RND)*K(I2,3)/H)
7630    IF H=0 THEN GOTO 7720
7640    LET S0=S0-H:GOSUB 7930:PRINT H;"UNIDADES ALCANZAN AL AVENTURA"
7650    LOCATE 15,44:PRINT "DESDE EL SECTOR";K(I2,1);K(I2,2)
7660    IF S0<0 THEN CLS:PRINT "EL AVENTURA HA SIDO DESTRUIDO":GOTO 8090
7670    LOCATE 16,44:PRINT "ESCUDO DISMINUYE A ":S0
7680    IF (H<20) OR (.02>=H/S0) OR (RND>.5) THEN GOTO 7720
7690    LET I=INT(1+RND*8): D(I)=D(I)-(H/S0)-.5*RND
7700    LOCATE 17,44:PRINT "CONTROL DE DA%OS INFORMA:"
7710    LOCATE 18,44:GOSUB 6080:PRINT DD$;"DANADO"
7720    GOSUB 8020
7730 NEXT I2
7740 RETURN
7750 REM
7760 REM *** REPARACIONES EN MARCHA ***
7770 REM
7780 LET D1=14:LET D6=W1:IF D6>1 THEN LET D6=1
7790 FOR I=1 TO 8
7800    IF D(I)=0 THEN GOTO 7850
7810    LET D(I)=D(I)+D6:IF D(I)<0 THEN GOTO 7850
7820    IF D(I)>0 THEN LET D(I)=0
7830    LOCATE D1,44:GOSUB 6080
7840    PRINT DD$;"REPARADO ":GOSUB 8020
7850 NEXT I
7860 IF RND>.1 THEN RETURN
7870 LET I=1+INT(8*RND)
7880 LET D(I)=D(I)-.1-RND*5
7890 GOSUB 6080
7900 GOSUB 7930:PRINT "CONTROL DE DANOS INFORMA:"
7910 LOCATE 15,44:PRINT DD$;" DANADO  "
7920 RETURN
7930 REM
```

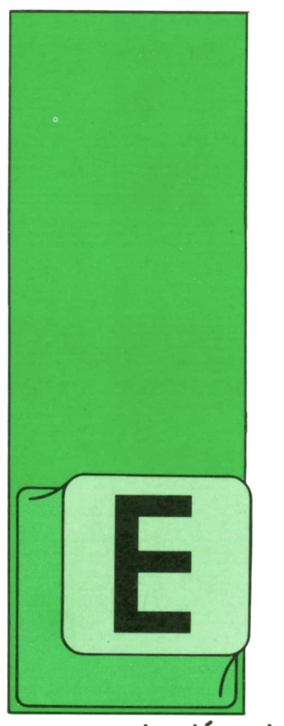
```
7940 REM *** BORRA ZONA DE INFORMACION ***
7950 REM
7960 FOR J=0 TO 9
7970    LOCATE 23-J.44
7980    PRINT "                                        "
7990 NEXT J
8000 LOCATE 14.44
8010 RETURN
8020 REM
8030 REM *** RETARDO ***
8040 REM
8050 FOR I=1 TO 500
8060    LET J=2.2*I
8070 NEXT I
8080 RETURN
8090 PRINT "ES LA FECHA ESTELAR ";T
8100 PRINT "QUEDAN";K9;"NAVES ENEMIGAS EN LA GALAXIA AL FINAL DE SU MISION."
8110 PRINT "-LA FEDERACION SERA CONQUISTADA!"
8120 END
```

Para utilizar este programa en el AMSTRAD hay que realizar los siguientes cambios:

```
AMSTRAD

1900 MODE 2
1910 RANDOMIZE TIME
2100 LET AS=1+INT(8*RND(1))
2180 LET Q1=1+INT(8*RND(1)):LET Q2=1+INT(8*RND(1))
2190 LET S1=1+INT(8*RND(1)):LET S2=1+INT(8*RND(1))
2230 LET TO=100*(20+INT(20*RND(1)))
2640 LOCATE 6,2:PEN 3
2650 PRINT "     PANTALLA DE CORTO ALCANCE     ":PEN 2
3710 LOCATE 60,4:PEN 3:PRINT "ALERTA ROJA        ":PEN 2
3720 REM
3760 LOCATE 60,4:PEN 3:PRINT "ALERTA AMARILLA   ":PEN 2
4040 REM
4170 REM
5160 LET C1=C1*RND(1)
5200 LET H=INT((2+RND(1)*C1/SQR(((K(I,1)-S1)*(K(I,1)-S1))
+((K(I,2)-S2)*(K(I,2)-S2))))
5620 LOCATE 6+XK,3+XH:PRINT " ":GOTO 5400
6230 LOCATE 1,24
6250 REM
6370 LOCATE 5,13:PRINT "      MAPA ACUMULADO DE LA GALAXIA"
6390 LOCATE 6,15:PRINT "====================================="
7620 LET H=INT((2+RND(1)*K(I2,3)/H)
7680 IF (H<20) OR (.02>=H/SO) OR (RND(1)>.5) THEN 7720
7690 LET I=INT(RND(1)*8+1):LET D(I)=D(I)-(H/SO)-.5*RND(1)
7860 IF RND(1)>.1 THEN RETURN
7870 LETR I=1+INT(8*RND(1))
7880 LET D(I)=D(I)-.1-RND(1)*5
```

TECNICAS DE ANALISIS



Bases de datos relacionales

EL modelo relacional ha surgido de la aplicación de la teoría matemática de las relaciones a los modelos jerárquicos, como optimización de los esquemas de representación de red. Desde el punto de vista práctico (y al nivel de visión general en que nos encontramos), las «relaciones» se pueden concebir como unas tablas que incorporan tanto las informaciones propias de cada concepto como las relaciones de dependencia existentes entre ellos.

Si, por ejemplo, hemos de manejar la información sobre nuestra biblioteca que ya conocemos:

WIRTH, Niklaus Aleman. Inst. Fed. Tecn.
Alg.+Estruc. datos=Programas 382 Casti 1980 1200 Díaz Sant.
The design of a Pascal Compiler 150 Amms. 1971 98 FF Lib. Univ.
HARTNELL, Tim Británico.
Int. Art.: conceptos y programas 267 Anaya 1985 1.500 Lib. Tecni.
Libro Gig. Jueg. para ZX Spectrum 310 Anaya 1986 1.750 Edic. Perg.
Libro Gig. Jueg. para ordenador 259 Anaya 1984 1.500 Lib. Tecni.
GRIES, David Estadouni. Cornell University.
Compiler Cnst. for Dig. Computers 452 JhonW 1984 32$$ Ther. Lib.

se podrían establecer las siguientes relaciones en el conjunto:

AUTORES

núm. autor	nombre	nacionalidad	institución
A1	WIRTH, Niklaus	alemana	Ins. Fed. Tecn.
A2	HARTNELL, Tim	británica	
A3	GRIES, David	estadounidense	Cornell Univers.

VOLUMENES

núm. volumen	título	pág.	ed.	año	precio	librero	autor
V1	Alg.+Estruc. datos=Programa	382	E3	1980	1200	L2	A1
V2	The Design of a Pascal Compilers	150	E1	1971	98FF	L4	A1
V3	Int. Art.: conceptos y programas	267	E2	1985	1500	L3	A2
V4	Libro Gig. Jueg. para ZX Spectrum	310	E2	1986	1750	L1	A2
V5	Libro Gig. Jueg. para ordenador	259	E2	1984	1500	L3	A2
V6	Complier Cnst. for Dig. Computer	452	E4	1984	32$$	L5	A3

EDITORIALES

núm. editorial	nombre	ciudad
E1	Editions Ammstensig	Londres
E2	Ediciones Anaya	Madrid
E3	Ediciones del Castillo	Madrid
E4	Jhon Wiley & Sons, Inc	N. Y.

LIBREROS		
núm. librero	nombre	ciudad
L1	Editorial Pergas	Madrid
L2	Díaz de Santos	Madrid
L3	Librería Técnica	Barcelona
L4	Librería Universitaria	Marsella
L5	Thecnical Librarians	Filadelfia

Como se ve, las relaciones propuestas incorporan informaciones sobre los diferentes conjuntos presentes (autores, volúmenes, editoriales y librerías), así como relaciones entre ellos (en VOLUMENES se relacionan los autores, las editoriales y los libreros), aunque podrían haberse suprimido las informaciones de correspondencia en la relación VOLUMENES y haber creado otras relaciones:

DE	
núm. autor	núm. volumen
A1	V1
A1	V2
A2	V3
A2	V4
A2	V5
A3	V6

COMPRADO EN	
núm. volumen	núm. librero
V1	L2
V2	L4
V3	L3
V4	L1
V5	L3
V6	L5

o, incluso, relaciones compuestas:

DATOS EDITORIALES		
núm. volumen	núm. editorial	núm. librero
V1	E3	L2
V2	E1	L4
V3	E2	L3
V4	E2	L1
V5	E2	L3
V6	E4	L5

La flexibilidad es grande y la facilidad para actualizar datos o relaciones, o para incluir relaciones nuevas, es también enorme.

La teoría de las relaciones se basa en diversas condiciones que se han de cumplir en las relaciones establecidas:

— No debe haber dos líneas iguales en una tabla.

— El orden de las líneas no es significativo.

- El orden de las columnas no es significativo.

Cada relación (tabla) tiene una clave de búsqueda que puede ser simple (un solo concepto) o compuesta de varias referencias.

La gran ventaja de estos SGBD's relacionales proviene de la facilidad de gestión que el álgebra relacional aporta, de tal modo que los sistemas resultan sumamente eficaces, a pesar de la redundancia de información que incluyen.



Lenguajes utilizados en los SGBD

Formando parte del propio SGBD, el administrador de la base dispone, usualmente, de varios lenguajes:

a) Lenguaje de definición de datos (DDI, Data Definition Language). Es el lenguaje utilizado para la definición del «esquema» —estructura general, ligazón de componentes, etc.— de la base.

b) Sub-lenguaje de definición de datos (DSDL, Data Subdefinition Language). Equivalente al DDL para la definición de los subesquemas que configuran la base de datos.

c) Lenguaje de manipulación de datos (DML, Data Manipulation Language). Lenguaje utilizado para la inclusión, modificación, extracción, ordenación, etc., de los datos presentes en la base.

d) Lenguaje de consulta o de búsque-

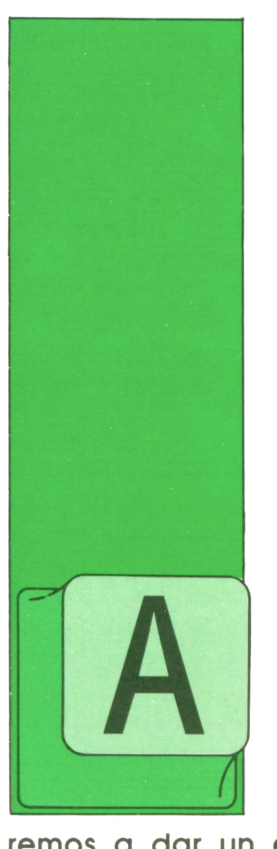
da (DIL, Data Inquisition Language). De características semejantes al anterior —DML—, se utiliza para establecer consultas a la base de un modo autónomo (bien consultas puntuales o búsquedas complejas bajo alguna condición).

Normalmente el lenguaje DML es manejado desde un lenguaje convencional de programación (COBOL, PL/1, ENSAMBLADOR, etc.): es decir, en el interior de un programa convencional se introduce un comando de DML para realizar la función deseada (almacenar un dato, leerlo, sustituirlo por otro, etc.), en vez de dar el comando correspondiente de manipulación de datos propio del lenguaje de que se trata: así, desde el programa se interactúa con la base de datos; pero en ocasiones es útil realizar una consulta en la base de datos o una búsqueda exhaustiva por uno o varios criterios de selección: en estos casos es útil que el SGBD disponga de un DIL para la consulta directa, sin necesidad de que deba hacerse un programa para una simple pregunta.

En algunos SGBD's se introduce un único lenguaje (SQL, Structured Query Language-Lenguaje de consultas estructuradas) que engloba todas las funciones antedescritas: definición de datos, consultas desde programas, consultas directas, especificación de restricciones, etc.

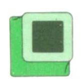
Respecto de los lenguajes de manipulación de datos, se suelen clasificar en tres grupos: lenguajes «de navegación», lenguajes algebraicos y lenguajes orientados al cálculo. Los primeros se utilizan básicamente en las bases de datos jerárquicas o en red y se caracterizan porque, en algún modo, en la propia consulta se define el camino de acceso a los datos, por su situación o por su posición en el conjunto. El lenguaje algebraico es el modo usual de gestión en las bases de datos relacionales: las consultas consisten en una secuencia de operaciones referidas a las relaciones presentes en la base e, incluso, la propia consulta es una relación. El último tipo se basa en el cálculo de predicados y en él desaparece la noción de secuencia: es un lenguaje no-procedural.

TECNICAS DE PROGRAMACION



Manejo de ficheros en PASCAL y APL

AL igual que vimos al hablar del manejo de la pantalla, en PASCAL y en APL el manejo de ficheros no está definido por completo de forma estándar, por lo que nos limitaremos a dar un ejemplo referido a un compilador o intérprete de cada uno de estos lenguajes. En el caso del primero, nos referiremos al compilador TURBO PASCAL, de la casa Borland. En el caso del APL tomaremos como ejemplo el intérprete APL/PC de IBM.



Operaciones con ficheros en PASCAL

Vamos a ver un ejemplo sencillo: un programa que crea un fichero y pide a la persona sentada ante el terminal que le proporcione mediante el teclado las líneas de texto que pasarán a formar los registros del fichero, y otro programa que lee el fichero creado por el programa anterior y escribe los registros en la pantalla. Las operaciones de ficheros que tendrá que utilizar el primer programa son las siguientes:

1. Declarar una variable de tipo fichero.
2. Asignar nombre al fichero.
3. Crear el fichero y prepararlo para su utilización.
4. Escribir líneas de texto en el fichero.
5. Cerrar el fichero.

Veamos el primer programa:

```
program ESCRIBIR;
var
  f:text;
  linea:array [1..80] of char;
  j:integer;
begin
  assign (f,'fichero');
  rewrite (f);
  writeln ('Dame los registros
   del fichero');
   repeat
    readln(linea);
    j:=80;
    while (j<>0) and (linea[j]=' ')
    do j:=j-1;
    if j<>0 then writeln(f,linea)
   until j=0;
   close(f)
 end.
```

Los puntos de interés de este programa son los siguientes:

1. Es preciso declarar una variable de tipo fichero de texto (f:text). La declaración se hace de un modo muy parecido al de una variable ordinaria (j:integer).
2. Es preciso asignarle a dicha variable el nombre del fichero correspondiente. Esto se consigue con una instrucción especial: ASSIGN(f,'fichero').
3. A continuación hay que abrir el fichero para escribir en él. Esto lo hace la instrucción: REWRITE(f).
4. El programa escribe ahora un mensaje por la pantalla ('Dame los registros del fichero'), pidiendo a la persona sentada al terminal que le proporcione diversas líneas de texto que deben incorporarse al fichero. Este proceso continuará hasta que se le proporcione al programa una línea vacía.
5. Dentro del bucle REPEAT se empieza por leer una línea y por eliminar de ella los espacios en blanco situados al final. Esto es lo que consigue el bucle WHILE que, además, calcula el número de caracteres de la línea (una vez eliminados los blancos finales) sobre la variable entera j.

6. Si j es distinto de cero (es decir, si la línea no estaba vacía), se escribe esta línea en el fichero como un nuevo registro. Esto lo hace la operación WRITELN (f, línea), que funciona de un modo totalmente análogo a la función WRITELN cuando se utiliza para escribir en la pantalla.

7. Cuando j es igual a cero (es decir, cuando le hemos dado una línea vacía), el bucle REPEAT-UNTIL llega a su fin. Entonces se cierra el fichero con la instrucción CLOSE(f) y el programa queda terminado.

Veamos un ejemplo de la utilización de este programa:

```
RUN
Dame los registros del fichero
registro 1
1 2 3
Este es el registro más largo de todos
¡HOLA!
```

Como se ve, hemos escrito cuatro registros de distinta longitud. La quinta línea la dimos en blanco (es decir, simplemente presionamos la tecla ENTER) y esto fue la señal para dar por terminado el programa.

Ahora vamos a pasar al segundo programa PASCAL, que lee el fichero construido por el programa anterior y lo escribe en la pantalla (cada registro en una línea diferente).

```
program LEER;
var
  f:text;
  linea:array [1..80] of char;
begin
  assign (f,'fichero');
  reset (f);
  writeln ('Los registros del
   fichero son:');
  while not eof(f) do begin
    readln(f,linea);
    write(linea)
    end;
  close(f)
end.
```

Veamos los principales puntos de interés de este programa:

1. Al igual que en el programa anterior, hay que declarar una variable de tipo fichero de texto (f:text).

2. También aquí es preciso asignarle a dicha variable el nombre del fichero correspondiente: ASSIGN(f,'fichero').

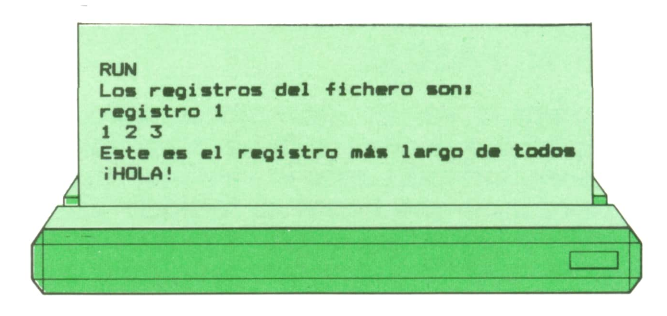
3. A continuación hay que abrir el fichero para leer de él. Esto lo hace la instrucción: RESET(f).

4. El programa escribe ahora un mensaje por la pantalla ('Los registros del fichero son:').

5. Dentro del bucle WHILE se empieza por leer una línea del fichero con la instrucción READLN(f,línea), que funciona igual que la instrucción READLN cuando se leen datos del teclado. A continuación, se escribe el registro leído en la pantalla con la instrucción WRITE(línea). Obsérvese que READLN, WRITELN, READ y WRITE se dirigen automáticamente al teclado o a la pantalla si no se especifica ningún fichero de trabajo.

6. El bucle WHILE continúa hasta que EOF(f) sea verdad. Esta es una función del sistema que nos dice si hemos encontrado la marca de fin de fichero (es decir, si se nos ha terminado el fichero). Por tanto, este programa leerá y escribirá por la pantalla todos los registros del fichero, hasta que se termine. En este momento se ejecutará la última instrucción, CLOSE(f), que cierra el fichero. El programa ha terminado.

Veamos cómo leería este programa el fichero creado por el programa anterior:



Operaciones con ficheros en APL

En el intérprete APL/PC de IBM existe un espacio de trabajo que contiene un conjunto de funciones básicas que pueden utilizarse para trabajar con ficheros. Estas funciones, que realizan operaciones parecidas a las descritas en BASIC y PASCAL, son las siguientes:

— WOPEN: Abre un fichero para lectura y escritura. El argumento izquierdo de esta función, si existe, es el número del fichero (si no existe, se supone que es 1). El argumento derecho es el nombre del fichero, seguido por una coma y la letra D (entre comillas).

— OPEN: Abre un fichero para lectura únicamente. Los argumentos son los mismos que los de WOPEN.

— WRITEV: Escribe un registro de texto (una línea) en un fichero abierto por WOPEN. El argumento izquierdo es el mismo que el de WOPEN. El argumento derecho es el texto a escribir en el fichero.

— READV: Lee el registro siguiente de un fichero abierto por OPEN o WOPEN. Su argumento derecho es el número del fichero.

— CLOSE: Cierra el fichero. Su argumento derecho es el número del fichero.

Veamos cómo se escribiría en APL el primer programa que sirvió como ejemplo en PASCAL, junto con un ejemplo de su utilización.

```
[0]  ESCRIBIR
[1]  1 WOPEN 'FICHERO,D'
[2]  'Dame los registros del fichero'
[2] BUCLE:A←⍞
[3]  →(0=⍴A)/FIN
[4]  1 WRITEV A
[5]  →BUCLE
[6] FIN:CLOSE 1
       ESCRIBIR
Dame los registros del fichero
registro 1
1 2 3
Este es el registro más largo de todos
¡HOLA!
```

Veamos ahora el segundo programa, que lee los registros del fichero creado por el programa anterior y los escribe por la pantalla, junto con su ejecución.

```
[0]  LEER
[1]  1 OPEN 'FICHERO,D'
[2]  'Los registros del fichero son:'
[2] BUCLE:A←READV 1
[3]  →(0=⍴A)/FIN
[4]  A
[5]  →BUCLE
[6] FIN:CLOSE 1
       LEER
```

```
Los registros del fichero son:
registro 1
1 2 3
Este es el registro más largo de todos
¡HOLA!
```

En APL es también posible (y muy sencillo) acceder directamente a la información contenida en los registros en un orden diferente al secuencial. En general, esto es útil cuando los registros tienen longitud fija (pues entonces el sistema puede calcular con facilidad dónde comienza el registro que le hemos pedido). Para realizar las operaciones de lectura y escritura directa disponemos de las mismas funciones indicadas anteriormente, aunque en lugar de WRITEV y READV utilizaremos las dos siguientes:

— WRITE: Escribe un registro de texto (una línea) en un fichero abierto por WOPEN. El argumento izquierdo es un conjunto de tres números: el primero es el número del fichero; el segundo el número del registro sobre el que queremos escribir; el tercero es la longitud del registro. Se supone que se trata de un fichero cuyos registros tienen todos la misma longitud. El argumento derecho es el texto a escribir en el fichero.

— READ: Lee un registro de un fichero abierto por OPEN o WOPEN. Su argumento derecho es igual que el argumento izquierdo de WRITE.

Si al abrir un fichero no ponemos la coma y la D detrás del nombre del fichero, el sistema interpretará que no se trata de un fichero de texto, sino que cada registro podrá contener información de cualquier tipo (números, caracteres, matrices, vectores, etc.). Esto permite una gran libertad en el almacenamiento de datos complejos en ficheros en disco.

Veamos un ejemplo en el siguiente programa, en el que los comentarios incluidos antes de cada línea explican su funcionamiento.

```
[0]  FICHERO
[1]  ⍝ Creación de un fichero
[2]   WOPEN 'F1'
[3]  ⍝ Inicializamos el contador
[4]   I←0
[5]  ⍝ Leemos una línea del terminal
[6]  L:'Dame un registro para el
     fichero'
```

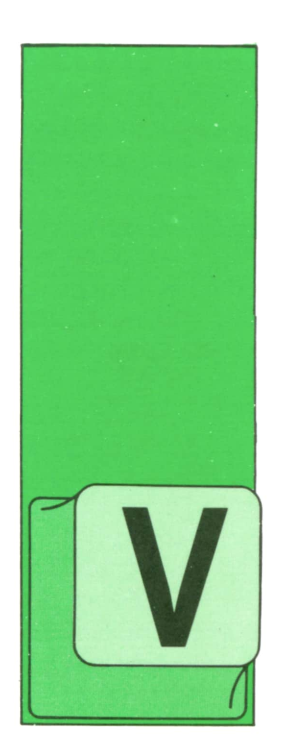
```
[7]    A←⎕
[8]   ⍝ Fin del fichero si está vacío
[9]    →(0=⍴A)/LEER
[10]  ⍝ Añadir un registro de 100 bytes
[11]   (1,I,100)WRITE A
[12]  ⍝ Buscar más registros
[13]   I←I+1
[14]   →L
[15]  ⍝ Ahora vamos a leer registros
[16]  LEER:'Dame el número del registro'
[17]   A←⎕
[18]  ⍝ Terminar si el número es negativo
[19]   →(A<0)/FIN
[20]  ⍝ O demasiado grande
[21]   →(A≥I)/FIN
[22]  ⍝ Leemos el registro y lo escribimos
[23]  ⍝ en la pantalla
[24]   READ 1,A,100
[25]  ⍝ Leemos más registros
[26]   →LEER
[27]  ⍝ Cerramos el fichero
[28]  FIN:CLOSE 1

      FICHERO
Dame un registro para el fichero
⎕:
      1 2 3 4 5 6
Dame un registro para el fichero
⎕:
      'ABCDEFGH'
Dame un registro para el fichero
⎕:
      3 4 ⍴ ⍳ 12
Dame un registro para el fichero
⎕:
      3 3 ⍴ 'ABCD'
Dame un registro para el fichero
⎕:
      ⍳ 10
Dame un registro para el fichero
⎕:
      0⍴0
```

```
Dame el número del registro
⎕:
      3
ABC
DAB
CDA
Dame el número del registro
⎕:
      1
ABCDEFGH
Dame el número del registro
⎕:
      0
1 2 3 4 5 6
Dame el número del registro
⎕:
      4
1 2 3 4 5 6 7 8 9 10
Dame el número del registro
⎕:
      2
  1   2   3   4
  5   6   7   8
  9  10  11  12
Dame el número del registro
⎕:
      1
ABCDEFGH
Dame el número del registro
⎕:
      1
ABCDEFGH
Dame el número del registro
⎕:
      10
```

LOGO

Otra manera de dar órdenes a la tortuga



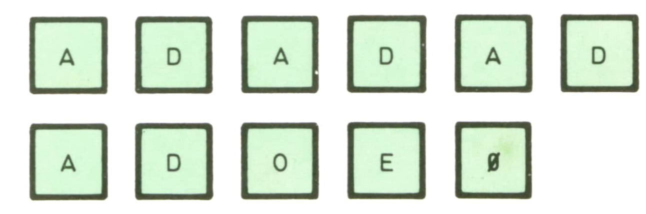
AMOS a definir un procedimiento que nos va a servir para mandarle a la tortuga que ejecute comandos, pero en lugar de escribirlos como hasta ahora utilizaremos una tecla para cada uno.

El procedimiento podría ser:

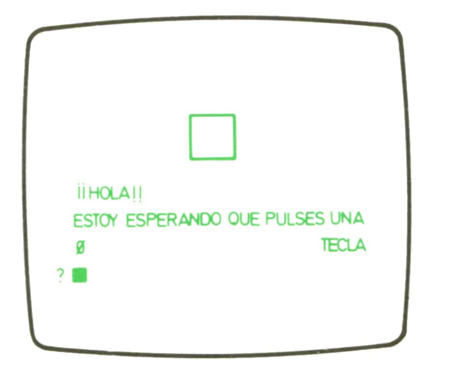
```
? PARA COMANDOS
> ES [ESTOY ESPERANDO QUE
PULSES UNA TECLA]
> HAZ "TECLA LC
> ES :TECLA
> SI :TECLA = 0 [ALTO]
> SI :TECLA = "A [AV 10]
> SI :TECLA = "R [RE 10]
> SI :TECLA = "D [GD 90]
> SI :TECLA = "I [GI 90]
> SI :TECLA = "L [BP]
> SI :TECLA = "S [SL]
> SI :TECLA = "B [BL]
> SI :TECLA = "O [OT]
> SI :TECLA = "M [MT]
> SI :TECLA = "E [ES "¡¡HOLA!!]
> SI :TECLA = "C [REPITE 36
  [AV 1 GD 10]]
> SI :TECLA = "G [GOMA]
> COMANDOS
> FIN
```



Ahora podemos hacer algunos dibujos pulsando teclas. Por ejemplo, si ejecutamos el procedimiento COMANDOS y pulsamos las siguientes teclas:



nos queda:



Podemos asignar un comando a todas las teclas que queramos teniendo cuidado de no usar una misma tecla para varias comandos.

Os proponemos

1. Escribe un procedimiento parecido al anterior, pero que en lugar de asignar comandos a las teclas éstas nos sirvan para hacer figuras. Por ejemplo:

T → triángulo
C → cuadrado
P → pentágono
R → círculo

.
.
.

Otro comando de escritura

Cuando queremos que la tortuga nos escriba algo en la pantalla utilizamos el comando ES. Pues bien, hay otro comando que también nos permite hacer esto, aunque de una forma un poco diferente.

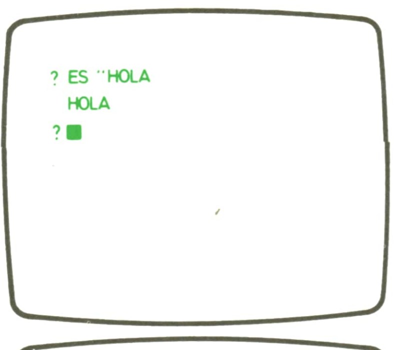
El comando

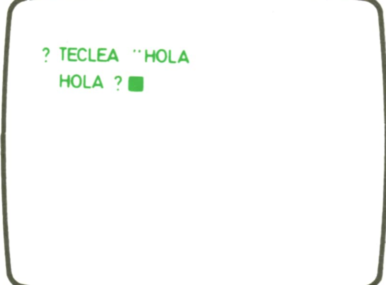
TECLEA obj

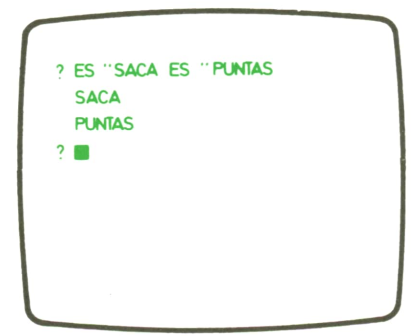
sirve para escribir **obj** (número, letra, palabra, lista o contenido de una variable)

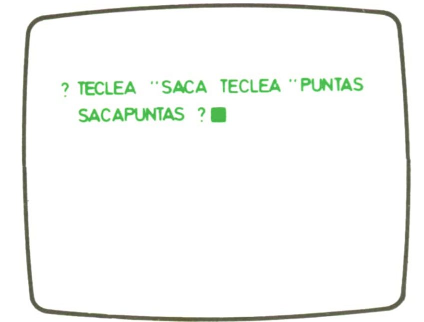
por la pantalla, pero sin saltar a la línea siguiente, mientras que ES sí que lo hace.

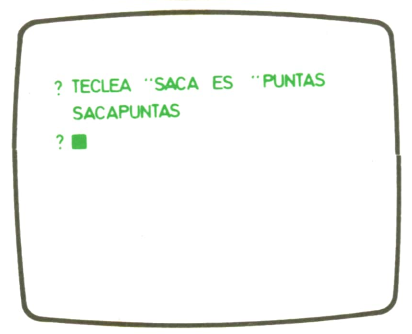
Vamos a ver la diferencia con algunos ejemplos:











Una función nueva

Puede haber casos en que nos interese saber si una determinada cosa forma parte de una lista.

Para eso podemos utilizar la función

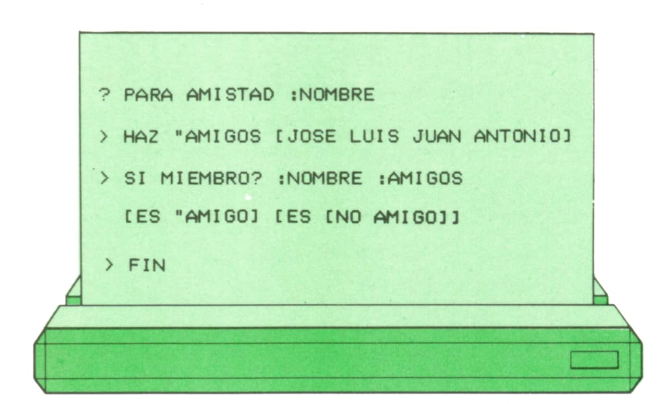
MIEMBRO? obj lista

que devuelve CIERTO si **obj** (número, letra, palabra...) está dentro de **lista** y FALSO si no es así.

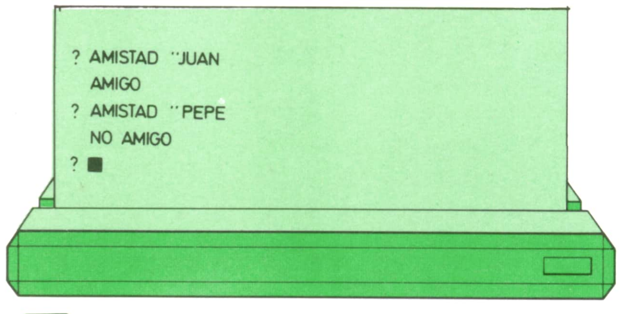
Como siempre, al ser MIEMBRO? una función tenemos que usar el resultado para algo. En este caso lo normal será hacerlo como condición para el comando SI.

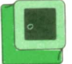
Por ejemplo, supongamos que guardamos en una variable los nombres de nuestros amigos y luego queremos saber si hemos metido o no a uno en concreto.

El procedimiento nos quedaría así:



y al ejecutarlo saldría:





Un procedimiento para jugar

El juego va a consistir en adivinar un número que la tortuga va a pensar. Para ello nuestra amiga calcula un número comprendido entre 0 y 9, aleatoriamente. Después, nosotros iremos pulsando teclas hasta que acertemos.

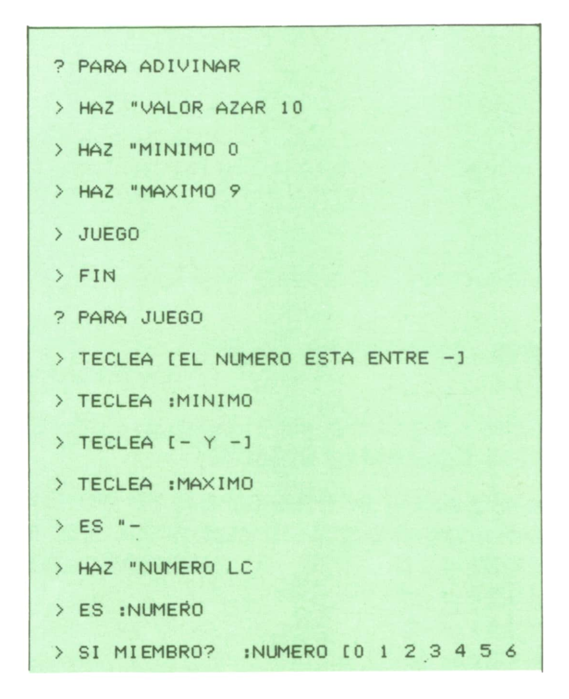
En el procedimiento vamos a utilizar varias variables:

— VALOR para guardar el número que la tortuga calcule.

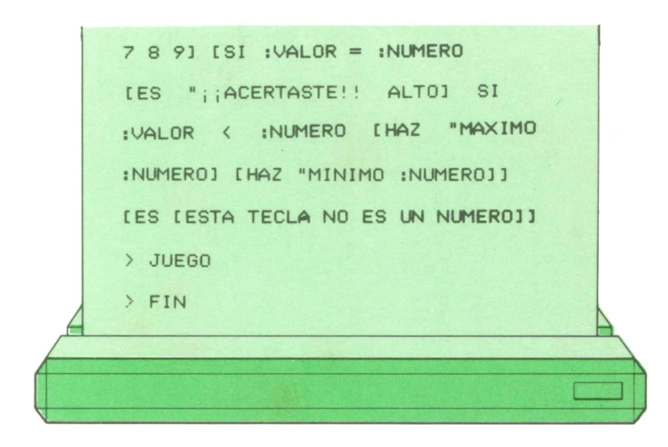
— NUMERO para ir guardando la tecla que pulsemos.

— MAXIMO y MINIMO para ir sabiendo entre qué números está comprendido el número que tenemos que acertar.

Tenemos que escribir:



```
? PARA ADIVINAR
> HAZ "VALOR AZAR 10
> HAZ "MINIMO 0
> HAZ "MAXIMO 9
> JUEGO
> FIN
? PARA JUEGO
> TECLEA [EL NUMERO ESTA ENTRE -]
> TECLEA :MINIMO
> TECLEA [- Y -]
> TECLEA :MAXIMO
> ES "-
> HAZ "NUMERO LC
> ES :NUMERO
> SI MIEMBRO? :NUMERO [0 1 2 3 4 5 6
```



```
7 8 9] [SI :VALOR = :NUMERO
[ES "¡¡ACERTASTE!! ALTO] SI
:VALOR < :NUMERO [HAZ "MAXIMO
:NUMERO] [HAZ "MINIMO :NUMERO]]
[ES [ESTA TECLA NO ES UN NUMERO]]
> JUEGO
> FIN
```

y para ejecutarlo pondremos:

?ADIVINAR

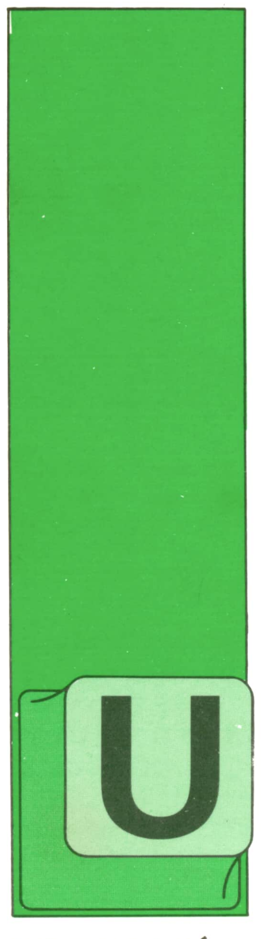


Os proponemos

1. Añade a los dos procedimientos anteriores los comandos y variables necesarias para que la tortuga te dé como máximo cinco oportunidades para acertar el número y si no lo logras que calcule otro nuevo.

Pista: Mientras no se hayan agotado las cinco oportunidades hay que llamar a JUEGO y cuando se alcance este tope a ADIVINAR.

PASCAL





El Pascal y las matemáticas

UNA de las aplicaciones más antiguas de los computadores es la resolución de problemas matemáticos. En estos casos, como en tantos otros, el empleo de lenguajes de programación estructurados, como el Pascal, puede simplificar mucho el desarrollo de programas. Para ilustrar esto vamos a ver cómo se puede operar con polinomios de manera muy cómoda.

Almacenamiento de polinomios en memoria

Para guardar un polinomio en memoria bastaría, en principio, con utilizar una tabla de reales donde almacenar sus coeficientes; sin embargo, y para facilitar los cálculos, también es interesante conocer el grado del polinomio, por lo que utilizaremos las siguientes definiciones:

```
const
  Grado_Max = 15; (* por poner un límite...*)
type
  Grado_t       = 0..Grado_Max;
  Polinomio_t = record
                      Coef : array [Grado_t]
                                 of real;
                      Grado: Grado_t
                    end;
```

Si el polinomio fuese, por ejemplo:

```
5 * X^3 + X^2 - 3 * X + 10
```

(donde ^ significa «elevado a»), tendríamos los siguientes valores:

```
Grado     =  3
Coef [3] =  5
Coef [2] =  1
Coef [1] = -3
Coef [0] = 10
```

El valor de los restantes coeficientes sería indiferente, al conocerse, gracias a Grado, el grado del polinomio.

Evaluación de un polinomio

Para conocer el valor de un polinomio en un punto dado, en lugar de hacer exponenciaciones, utilizaremos el método de Horner, que es mucho más eficiente. Tomando como ejemplo un polinomio de cuarto grado, consistiría en hacer la siguiente transformación:

```
A * X^4 + B * X^3 + C * X^2 + D * X + E
```

es igual a

```
(((A * X + B) * X + C) * X + D) * X + E
```

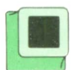
con lo que el cálculo se limita a hacer unas pocas multiplicaciones y sumas. El procedimiento de evaluación podría ser:

```
function VALOR (var P: Polinomio_t; X: real): real;
{---------------------------------------}
{  Da el valor del polinomio P en X  }
{---------------------------------------}
 var Acumula: real; I: integer;
begin
 Acumula := 0.0;
 with P do
   for I := Grado downto 0 do
     Acumula := Acumula * X + Coef [i];
 Valor := Acumula
end;
```

Con esta función podríamos escribir, por ejemplo:

```
B:= Valor (Pol,X);
```

El paso de parámetro se hace por nombre en el caso del polinomio para ahorrar tiempo.



Derivada de un polinomio

La obtención a partir de un polinomio dado del polinomio que es su derivada es muy sencilla; tomando como ejemplo nuevamente el polinomio de cuarto grado anterior, su derivada sería:

4 * A * X^3 + 3 * B * X^2 + 2 * C * X + D

El procedimiento podría ser:

```
procedure DERIVAR (var P, D: Polinomio-t);
---------------------------------------
    Hace D igual a la derivada de P
---------------------------------------
   var I: integer;
begin
 for I := 1 to P. Grado do
    D.Coef (i-1):= P.Coef (i) * I;

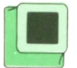
 D.Grado := P.Grado - 1
end;
```

En este caso, el paso de D por nombre es obligatorio. Se podría acelerar algo el proceso poniendo:

```
begin
with P do
   begin
   for I := 1 to Grado do
      D.Coef (i-1) := Coef (i) * I;
   D.Grado := Grado - 1
   end
end;
```

El hecho de utilizar un bucle con valores crecientes de I permite convertir a un polinomio en su derivada directamente:

```
Derivar (A,A);
```



Normalización de polinomios

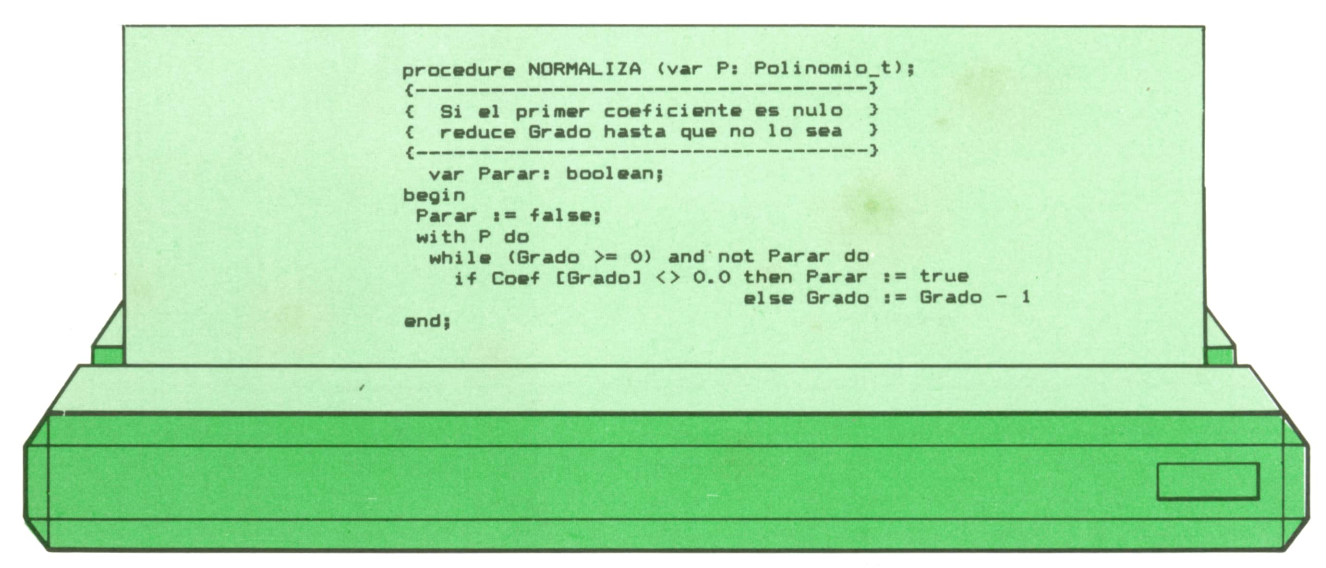
Con la estructura de datos utilizada, un mismo polinomio puede, en principio, tener varias representaciones distintas; el polinomio de grado 1:

A * X + B

es igual al polinomio de grado 2:

0 * X^2 + A * X + B

por tanto, para aprovechar la ventaja de almacenar el grado y para simplificar el procedimiento de división, como se verá más adelante, conviene «normalizar» la representación del polinomio, es decir, reducir su grado para eliminar los coeficientes nulos superfluos:



```
procedure NORMALIZA (var P: Polinomio_t);
{--------------------------------------}
{  Si el primer coeficiente es nulo   }
{  reduce Grado hasta que no lo sea   }
{--------------------------------------}
  var Parar: boolean;
begin
 Parar := false;
 with P do
  while (Grado >= 0) and not Parar do
    if Coef [Grado] <> 0.0 then Parar := true
                           else Grado := Grado - 1
end;
```

Este procedimiento tiene el inconveniente de que las condiciones de salida del bucle son la llegada al primer coeficiente no nulo o que Grado valga -1; Grado es un campo de tipo Grado-t y, por tanto, no puede tomar ese valor. La solución más sencilla consiste, pues, en definir Grado como de tipo Integer.



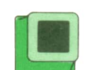
División de polinomios

Para dividir polinomios hay que restar al dividendo el divisor multiplicado por un término tal que el primer elemento de aquél se anule, y seguir haciendo lo mismo con el resto obtenido hasta que éste sea de grado menor que el divisor; el conjunto de términos utilizados es, precisamente, el cociente de la división.

```
procedure DIVIDIR (var N, D, C, R: Polinomio_t):
{-----------------------------------}
{  Hace C y R igual al cociente y   }
{  al resto de dividir N entre D.   }
{-----------------------------------}
 var K: real; I, Dif: integer;
begin
Normaliza (D);    { Por si su primer coeficiente es 0. }
if D.Grado >= 0 then  { Si es un polinomio no nulo... }
    begin
    R := N;
    C.Grado := N.Grado - D.Grado;
    with R do
     while Grado >= D.Grado do
      begin
       { exponente y coeficiente a utilizar:  }
       Dif := Grado - D.Grado:
       K   := Coef [Grado] / D.Coef [D.Grado];

       { Resto := Resto - (K * X^Dif) * Divisor:  }
       for I := Grado - 1 downto Dif do
            Coef[i]:= Coef[i] - K * D.Coef[i-dif];
       Grado := Grado - 1;

       { el término empleado corresponde al cociente: }
       C.Coef [Dif] := K
      end
    end
  else
    begin
     writeln;
     writeln ('Error:  División por polinomio nulo.');
     writeln ('Pulse Intro para seguir.');
     readln
    end
end;
```



Producto de polinomios

El producto de polinomios se implementa de una manera muy sencilla: basta con utilizar un bucle para ir recorriendo los términos de uno de los polinomios y, con cada uno de ellos, utilizar otro bucle para irlo multiplicando por todos los términos del otro:

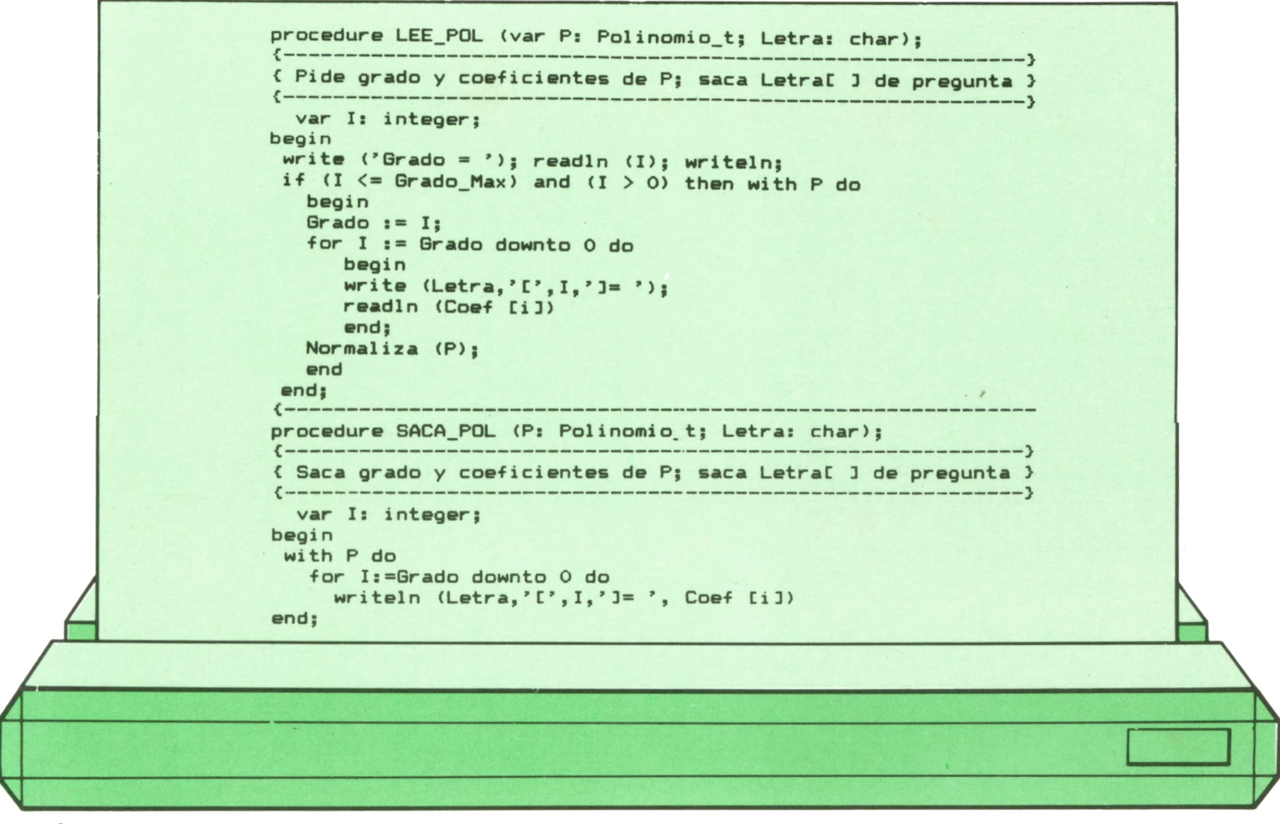
```
procedure MULTIPLICAR (var P, Q, Res: Polinomio_t);
{------------------------------}
{  Hace Res igual al producto }
{  producto de P por Q.       }
{------------------------------}
 var I,J: integer;
begin
 Normaliza (P);
 Normaliza (Q);
 if P.Grado + Q.Grado > Grado_Max then
   writeln ('El resultado es de un grado excesivo.')

else with Res do
  begin
   Grado := P.Grado + Q.Grado;
   for I := 0 to Grado do Coef [i] := 0.0;

   for I := 0 to P.Grado do
     for J := 0 to Q.Grado do
       Coef [i+j] := Coef [i+j] + P.Coef [i] * Q.Coef [j]
  end
end;
```

Introducción y presentación de datos

Para terminar, veamos cómo introducir los coeficientes de un polinimio desde teclado y cómo presentarlos en pantalla:

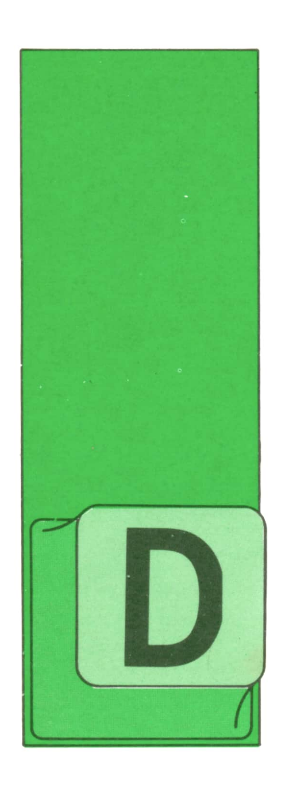


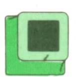
```
procedure LEE_POL (var P: Polinomio_t; Letra: char);
{-------------------------------------------------------------}
{ Pide grado y coeficientes de P; saca Letra[ ] de pregunta }
{-------------------------------------------------------------}
  var I: integer;
begin
 write ('Grado = '); readln (I); writeln;
 if (I <= Grado_Max) and (I > 0) then with P do
   begin
   Grado := I;
   for I := Grado downto 0 do
      begin
      write (Letra,'[',I,']= ');
      readln (Coef [i])
      end;
   Normaliza (P);
   end
 end;
{-------------------------------------------------------------
procedure SACA_POL (P: Polinomio_t; Letra: char);
{-------------------------------------------------------------}
{ Saca grado y coeficientes de P; saca Letra[ ] de pregunta }
{-------------------------------------------------------------}
  var I: integer;
begin
 with P do
   for I:=Grado downto 0 do
     writeln (Letra,'[',I,']= ', Coef [i])
end;
```

Próximamente construiremos un programa que utilizará los procedimientos y funciones aquí desarrollados.

OTROS LENGUAJES

LISP (III)





El lenguaje LISP

DESPUES de hablar de las enormes posibilidades que el LISP ofrece, a pesar de disponer de un solo tipo de instrucción y de una estructura de datos tan sencilla, adentrémonos en el problema principal: la realización de un programa y las funciones esenciales para que esto sea realizable.

Pero antes demos una lista en la que se puede ver una breve definición de las funciones estándar del LISP, que prácticamente definen el lenguaje. Puede observarse que las principales no están descritas, ya que dejamos esta explicación para el texto.

Funciones más comunes en LISP

Notación utilizada en la definición de las funciones LISP:

A.....ATOM → Atomo
B.....Cuerpo del programa (Body)
F.....FUNCTION → Función
L.....LIST → Lista
NIL...Falso (Nada)
S.....S-expresión → Expresión simbólica
T.....Verdad (True)
X.....Número
AR....ARGUMENT → Argumento (Cualquiera de los anteriores)

En la notación se numera de la forma siguiente: 1 es la primera posición, 2 es la segunda en la lista, y 1...n se utiliza para cualquier número de argumentos de la clase especificada por la letra precedente.

FUNCION	Definición	Ejemplo
ABS	Devuelve el valor absoluto de su argumento.	(ABS X)
ADD1	Devuelve el argumento más uno.	(ADD1 X)
AND	Devuelve NIL si alguna de las S-expresiones es NIL y devuelve T en caso contrario.	(AND S...S)
APPEND	Devuelve una lista sencilla formada por los elementos de dos listas que son sus argumentos.	(APPEND L1 L2)
APPLY	Opera con los argumentos con la función dada como si apareciese primero en la S-expresión.	(APPLY F A)
ASSOC	Usa el primer argumento como clave y busca el mismo en la lista que figura como segundo argumento.	(ASSOC A L)
ATOM	Devuelve T si la s-expresión es un ATOM, si no devuelve NIL.	(ATOM S)
CAR	Devuelve el primer elemento dé la lista.	(CAR L)
CDR	Devuelve la lista sin el primer elemento.	(CDR L)
COND	Es la función de bifurcación del LISP.	(COND L1 L2)
CONS	Añade la s-expresión a la lista dada como primer elemento.	(CONS S L)
DEFINE	Define una función; se devuelve el nombre de la función.	(DEFINE F AR ...ARn...B)
DEFPROP	Instala la s-expresión como propiedad del átomo.	(DEFPROP A S)
DELETE	Elimina las veces que aparece la s-expresión en la lista. El número de borrados lo determina el tercer argumento. Si no	(DELETE S L X)

FUNCION	Definición	Ejemplo
	existe, se suprimen todas las coincidencias.	
DIFFE-RENCE	Devuelve el primer argumento menos el segundo.	(DIFFERENCE X1 X2)
DO	Es la función de iteración en LISP.	DO
EQUAL	Devuelve T si las s-expresiones son idénticas.	(EQUAL S S)
EVAL	Devuelve el valor de la evaluación de la s-expresión	(EVAL S)
EXPLODE	Devuelve una lista de un carácter formada con el carácter dado.	(EXPLODE A)
EXP	Eleva *e* a la potencia indicada.	(EXPO X)
EXPT	Eleva el primer argumento a la potencia indicada por el segundo argumento.	(EXPT X1 X2)
FUNCALL	Opera sobre los argumentos con la función dada (parecido a APPLY).	(FUNCALL F ARL...ARn)
LESSP	Devuelve T si todos los argumentos van en orden creciente.	(LESSP X1...Xn)
LIST	Devuelve una lista de *n* elementos construida con ellos.	(LIST S1...Sn)
MAPCAR	Aplica la función a la lista o listas de argumentos. Devuelve una lista de resultados.	(MAPCAR F AR1...ARn)
MAX	Devuelve el argumento mayor.	(MAX X1...Xn)
MEMBER	Devuelve T si la s-expresión es igual al elemento de nivel mayor (nivel 0) de la lista.	(MEMBER S L)
MIN	Devuelve el argumento menor.	(MIN X1...Xn)
GET	Devuelve el valor de la propiedad asociada al átomo.	(GET A P)
GO	Bifurcación incondicional.	(GO T)
GREA-TERP	Devuelve T si todos los argumentos están en orden creciente.	(GREATERP X1...Xn)
IMPLODE	Transforma el carácter dado en la lista en átomo.	(IMPLODE L)
LAMBDA	Similar a DEFINE.	(LAMBDA AR)
LAST	Devuelve la lista con todos los elementos, excepto el último.	(LAST L)
LENGTH	Devuelve el número de elementos de la lista.	(LENGTH L)
MINUS	Devuelve el argumento decrementado en uno.	(MINUS X)
MINUSP	Devuelve T si el argumento es negativo.	(MINUSP X)
NOT	Devuelve T si la s-expresión es NIL; en otro caso, devuelve NIL.	(NOT S)
NULL	Devuelve T si la s-expresión es una lista vacía o NIL; en otro caso, devuelve NIL.	(NULL S)
NUMBERP	Devuelve T si el argumento es un número.	(NUMBERP S)
OR	Devuelve T si al menos una s-expresión de sus argumentos es no-NIL. De lo contrario, devuelve NIL.	(OR S1...Sn)
PLUS	Suma de todos los argumentos.	(PLUS X1...Xn)
PLUSP	Devuelve T si el argumento es positivo.	(PLUSP X)
PRINC	Igual que PRINT, pero imprime un CRETURN antes y un espacio después.	(PRINC S)
PRINT	Imprime la s-expresión. Su valor es siempre T.	(PRINT S)
PROG	Crea variables y soporta iteraciones (ver texto).	* * * *
PUTPROP	Instala la s-expresión como propiedad de un átomo.	(PUTPROP A S)
QUOTE	Idéntico a 's-expresión.	(QUOTE S)
QUOTIENT	(QUOTIENT X1 X2)	
READ	Lee un valor que introduce en la lista.	(READ L)
REVERSE	Invierte el orden de los elementos de la lista.	(REVERSE L)
RETURN	Causa la devolución de la s-expresión como valor del PROG en el que aparece.	(RETURN S)
SET	Devuelve el segundo argumento al que asigna el átomo resultante de la evolución del primer argumento.	(SET S1 S2)
SETQ	Devuelve la s-expresión con el valor del átomo.	(SETQ A S)
SQRT	Devuelve la raíz cuadrada del argumento.	(SQRT X)
SUB1	Sustituye por la s-expresión1 todas las coincidencias de la s-expresión2 en la s-expresión3.	(SUBST S1 S2 S3)
TERPRI	Causa un retorno de carro.	(TERPRI)
TIMES	Devuelve el producto de los argumentos.	(TIMES X1...Xn)
ZEROP	Devuelve T si el argumento es cero.	(ZEROP X)

Expliquemos estas instrucciones por orden alfabético. La primera instrucción importante es la de bifurcación COND. La función COND es una lista todo lo larga que deseemos formada por la palabra clave (que es un átomo) COND, seguida por pares de elementos. El primero de cada par es la función de condición; si su valor es T, entonces se devuelve como valor de la función el que tiene el segundo elemento, que es ejecutado. En caso contrario, se pasa al segundo par (algo

así como un ELSE IF), y así sucesivamente hasta el último par, que de ser falsa su condición, obliga a la función a devolver NIL, y produce un error de ejecución. Si no se desea la detención del programa, la solución evidente es poner al final de la función una condición que se cumpla siempre (T) y como función un valor a devolver que nos indique esta situación.

Esta función permite cualquier tipo de combinación lógica de las que se realizan en programación clásica y algunas bastante difíciles de implementar en otros lenguajes.

La segunda función importante es DEFINE, ya que sirve para definir funciones nuevas a partir de otras preexistentes. Puede parecer que esta situación sea una limitación, pero de hecho, con las funciones estándar de LISP y haciendo definiciones recursivas, se puede definir todo (el problema es tener una idea clara de cómo hacerlo).

Esta función está formada también por un número infinito de pares. El primer elemento es el nombre de la nueva función, el segundo es la nueva función, que es una expresión formada por otras funciones.

La función LAMBDA, junto con la DEFINE, forman base de la construcción de nuevas funciones.

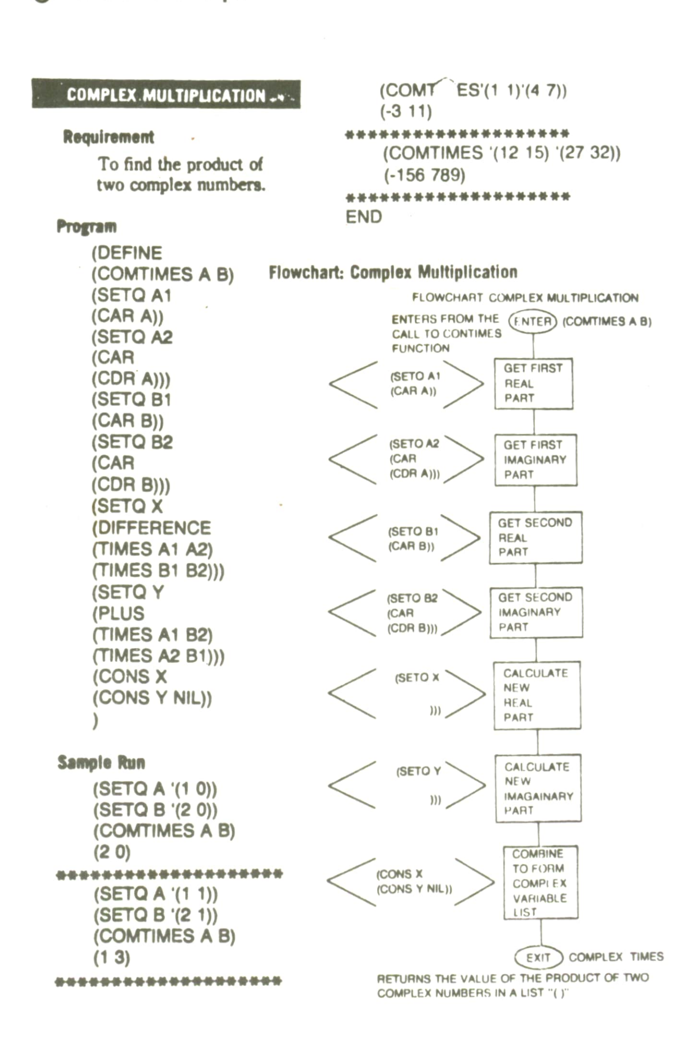
Por último, hablemos de la función principal: la función PROG. Esta función es la utilizada para formar programas con una «esencia» que tiene reminiscencias clásicas, ya que esta función permite el uso de declaraciones, sentencias, etiquetas, transferencias, funciones (sentencias) condicionales, y el retorno de valores.

Dentro de esta complicada función que llamamos PROG (programa), es decir, dentro de la lista que lo forma, hay sublistas, funciones, en definitiva, parámetros que cumplen estos cometidos.

Por ejemplo, las etiquetas son átomos, y las transferencias (los clásicos GOTO) son funciones que pueden estar en cualquier parte y que transfieren el control a otra.

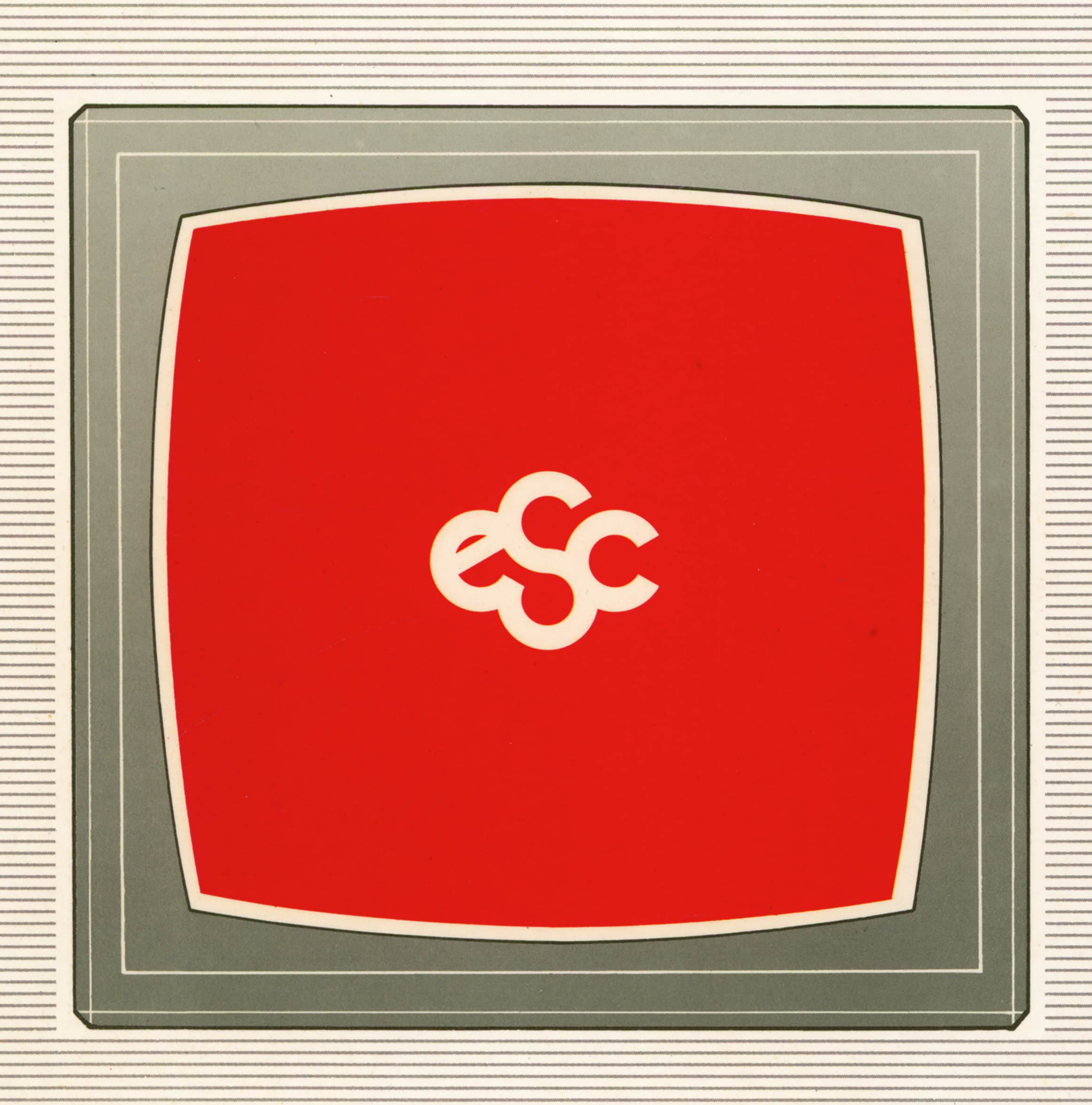
Evidentemente, las sentencias son una red de llamadas y retornos de funciones que cumplen el cometido del clásico cuerpo del programa.

La función condicional podría ser la imagen de la sentencia comparativa, y el retorno de valores es una función cuyo único cometido es transferir valores fuera de un determinado contexto. A continuación se puede ver un ejemplo de programa LISP típico:



Como resumen podríamos decir que no es una casualidad que el lenguaje LISP se imponga rápidamente en ambientes dedicados a la programación «a gran escala», incluso para actividades no relacionadas con la inteligencia artificial. Aunque si miramos la situación con una cierta perspectiva, qué software no será inteligencia artificial dentro de algunos años.

El programador imaginativo encontrará en este lenguaje una herramienta no sólo potente, sino de gran belleza conceptual. Sólo un planteamiento diferente al Funcional del LISP podrá competir en los lenguajes de programación futuros, y es la programación orientada a objetos. El LISP nos hace entrever un futuro lleno de sorprendentes cambios en el ámbito de la programación.



▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼